Optimisation dynamique de réseaux IP/MPLS

Josselin Vallet

Université Fédérale

Toulouse Midi-Pyrénées

THÈSE

En vue de l'obtention du

DOCTORAT DE L'UNIVERSITÉ DE TOULOUSE

Délivré par :

Institut National des Sciences Appliquées de Toulouse (INSA de Toulouse)

Présentée et soutenue par :
Josselin VALLET

le 05/05/15

Titre :

Optimisation dynamique des réseaux IP/MPLS

École doctorale et discipline ou spécialité :
EDSYS : Informatique 4200018

Unité de recherche :
Laboratoire d'Analyse et d'Architecture des Systèmes (LAAS-CNRS)

Directeur/trice(s) de Thèse :

Olivier BRUN et Balakrishna J. PRABHU

Jury :

Thierry GAYRAUD	Professeur à l'UPS	Président
Bernard FORTZ	Professeur à l'ULB	Rapporteur
Sandrine VATON	Professeur à Télécom Bretagne	Rapporteur
Jean-Marie GARCIA	Directeur de recherche CNRS	Examinateur
Eric GOURDIN	Chercheur à Orange Labs	Examinateur
Congduc PHAM	Professeur à l'UPPA	Examinateur
Olivier BRUN	Chargé de recherche CNRS	Directeur de thèse
Balakrishna J. PRABHU	Chargé de recherche CNRS	Directeur de thèse

Remerciements

Ce mémoire est l'aboutissement de trois années d'études effectuées au Laboratoire d'Analyse et d'Architecture des Systèmes du CNRS (LAAS-CNRS) au sein du groupe Services et Architectures pour Réseaux Avancés (SARA). Je tiens à exprimer toute ma reconnaissance à Jean ARLAT, directeur du LAAS, pour m'avoir permis de réaliser l'ensemble de ces travaux dans d'excellentes conditions.

J'adresse tous mes remerciements à mes deux directeurs de thèse, Monsieur Olivier BRUN et Monsieur Balakrishna PRABHU. Je les remercie tout particulièrement pour la qualité de leurs conseils et de leur encadrement. Ils ont su me guider avec patience, tout en me laissant une grande liberté dans la conduite des travaux de thèse. Leur grande disponibilité, leur dynamisme et leur bonne humeur m'ont fourni un cadre de travail particulièrement agréable. Je veux leur témoigner tout mon respect et mon amitié.

Je tiens à remercier l'ensemble des membres du jury pour avoir accepté d'évaluer la qualité de mes travaux. Tout d'abord Monsieur Thierry GAYRAUD, Professeur à l'Université Paul Sabatier de Toulouse, pour avoir accepté de présider le jury de cette thèse. Je suis tout particulièrement reconnaissant envers Monsieur Bernard FORTZ, professeur à l'Université Libre de Bruxelles, et Madame Sandrine VATON, professeur à Télécom Bretagne, qui m'ont fait l'honneur d'être les rapporteurs de cette thèse. Je les remercie chaleureusement pour leurs rapports et suggestions qui m'ont permis de parfaire ce mémoire. Je remercie également Monsieur Jean-Marie GARCIA, Monsieur Eric GOURDIN et Monsieur Congduc PHAM en leurs qualités de membre du Jury.

Je remercie tous les membres du LAAS, pour leur accueil, et particulièrement les nombreux amis que j'y ai rencontré. À commencer par l'ensemble des membres du bureau A45, avec lesquels j'ai pu passer d'excellents moments, aussi bien au LAAS qu'en dehors : Josu, Henda, Tatiana et Christopher. Merci à vous quatre. Merci aussi à tous les autres doctorants, membres de l'équipe SARA ou de QoS Design, avec qui j'ai pu partager de très agréables instants : Tom, Maialen, Ane, Samir, Urtzi, Cédric, Ahmad, Anouar, Sami. Je tiens aussi à remercier tout particulièrement Monsieur Hassan HASSAN, Ingénieur de Recherche au LAAS, avec lequel j'ai beaucoup apprécié de travailler

et de discuter.

Enfin, il me serait impossible de terminer ces remerciements sans évoquer ma famille et son soutien indéfectible. Je tiens à leur dédier cette thèse : à mes parents, à mon grand frère adoré et à sa compagne, à mes oncles et tantes, à mes grands-mères, mais aussi à mes grands-pères qui auraient été si fiers. Quel plaisir de partager des moments avec vous ! A tous, merci, merci, merci !

Josselin VALLET, mai 2015.

Résumé

La forte variabilité des trafics est devenue l'un des problèmes majeurs auxquels doivent faire face les gestionnaires d'infrastructures réseau. Dans ces conditions, l'optimisation du routage des flux en se basant uniquement sur une matrice de trafic moyenne estimée en heure de pointe n'est plus pertinente. Les travaux conduits dans cette thèse visent la conception de méthodes d'optimisation dynamiques du routage, adaptant en temps réel les routes utilisées par les flux aux conditions de trafic dans le réseau.

Nous étudions tout d'abord le problème d'optimisation des poids OSPF pour le routage intra-domaine dans les réseaux IP, où le trafic est routé le long de plus courts chemins, en fonction des poids des liens. Nous proposons une approche en ligne permettant de reconfigurer dynamiquement les poids OSPF, et donc les routes utilisées, pour répondre aux variations observées du trafic et réduire ainsi le taux de congestion du réseau. L'approche proposée repose sur l'estimation robuste des demandes en trafic des flux à partir de mesures SNMP sur la charge des liens. Les résultats expérimentaux, aussi bien sur des trafics simulés que réels, montrent que le taux de congestion du réseau peut être significativement réduit par rapport à une configuration statique.

Dans la même optique, nous nous intéressons également à l'optimisation des réseaux MPLS, qui permettent de gérer l'utilisation des ressources disponibles en affectant un chemin spécifique à chaque LSP. Nous proposons un algorithme inspiré de la théorie des jeux pour déterminer le placement des LSP optimisant un critère de performance non linéaire. Nous établissons la convergence de cet algorithme et obtenons des bornes sur son facteur d'approximation pour plusieurs fonctions de coût. L'intérêt principal de cette technique étant d'offrir des solutions de bonne qualité en des temps de calcul extrêmement réduits, nous étudions son utilisation pour la reconfiguration dynamique du placement des LSP.

La dernière partie de cette thèse est consacrée à la conception et au développement d'une solution logicielle permettant le déploiement d'un réseau overlay auto-guérissant et auto-optimisant entre différentes plateformes de cloud computing. La solution est conçue pour ne nécessiter aucun changement des applications. En mesurant régulièrement la qualité des liens Internet entre les centres de données, elle permet de détecter rapidement la panne d'une route IP et de basculer le trafic sur un chemin de secours. Elle permet également de découvrir dynamiquement les chemins dans le réseau overlay

qui optimisent une métrique de routage spécifique à l'application. Nous décrivons l'architecture et l'implémentation du système, ainsi que les expériences réalisées à la fois en émulation et sur une plateforme réelle composée de plusieurs centres de données situés dans différents pays.

Abstract

The high variability of traffic has become one of the major problems faced by network infrastructure managers . Under these conditions, flow route optimization based solely on an average busy hour traffic matrix is no longer relevant. The work done in this thesis aims to design dynamic routing optimization methods, adapting in real time the routes used by the flows to the actual network traffic conditions.

We first study the problem of OSPF weight optimization for intra-domain routing in IP networks, where the traffic is routed along shortest paths, according to links weights. We propose an online scheme to dynamically reconfigure the OSPF weights and therefore the routes used, to respond to observed traffic variations and reduce the network congestion rate. The proposed approach is based on robust estimation of flow traffic demands from SNMP measurements on links loads. Experimental results, both on simulated and real traffic data show that the network congestion rate can be significantly reduced in comparison to a static weight configuration.

On the same idea, we are also interested in optimizing MPLS networks that manage the available resource utilization by assigning a specific path for each LSP. We propose an algorithm inspired by game theory to determine the LSP placement optimizing a non-linear performance criterion. We establish the convergence of the algorithm and obtain bounds on its approximation factor for several cost functions. As the main advantage of this technique is to offer good quality solutions in extremely reduced computation times, we are studying its use for dynamic reconfiguration of the LSP placement.

The last part of this thesis is devoted to the design and development of a software solution for the deployment of a self-healing and self-optimizing network overlay between different cloud platforms. The solution is designed such that no change is required for client applications. By regularly measuring the quality of Internet links between data centers, it can quickly detect an IP route failure and switch the traffic to a backup path. It also allows to dynamically discover the paths in the overlay network that optimize a routing metric specific to the application. We describe the system architecture and implementation, as well as the experiments in both emulation and real platform composed of several data centers located in different countries.

Table des matières

Table des figures

1 Introduction

Internet constitue aujourd'hui une architecture de communication globale supportant une grande diversité de services, aussi bien professionnels que grand public. On assiste depuis plusieurs années à l'émergence et à la démocratisation rapide d'un ensemble de nouveaux usages : réseaux sociaux, streaming audio/vidéo, services en lignes sur plateformes de cloud, ... Les moyens d'accès au réseau continuent à s'améliorer et se diversifier, avec des réseaux filaires toujours plus rapides et des réseaux sans fils de plus en plus omniprésents.

Ces nouveaux usages et moyens d'accès entraînent une croissance exponentielle du nombre d'utilisateurs et de la quantité de données que doit transporter le réseau. Cette croissance met les fournisseurs d'accès et de services devant de réelles difficultés de gestion de ressources. Ces dernières sont en effet limitées mais doivent pouvoir faire transiter les trafics générés par les utilisateurs.

Un accroissement aveugle des capacités réseau pour répondre à l'augmentation du trafic représenterait un coût d'investissement considérable pour les opérateurs. Ces derniers ne disposent pas d'un budget illimité pour la mise à jour de leurs infrastructures, et ils cherchent donc à rentabiliser l'utilisation des ressources existantes. Ils doivent pour cela trouver un compromis entre 3 critères distincts et potentiellement antagonistes :

— minimiser les coûts de fonctionnement et de mise à jour de leurs infrastructures
— optimiser l'utilisation des ressources déjà existantes
— assurer un certain niveau de qualité de services à leurs clients.

La rentabilisation des ressources existantes passe nécessairement par une optimisation du routage des trafics dans le réseau, pour éviter les phénomènes de congestion et

de dégradation de services. Une optimisation efficace repose sur une bonne connaissance des infrastructures réseaux et des trafics qui y transitent. Les opérateurs réseau n'ont pas une connaissance exhaustive des trafics qui transitent sur leurs réseaux, et doivent faire appel à des techniques de métrologie réseau pour obtenir des estimations d'utilisation des ressources et qualité de service offerte, et adapter la configuration en conséquence.

Mais la grande volatilité des trafics d'aujourd'hui rend cette tâche particulièrement ardue : une configuration optimale à un instant de la journée peut rapidement devenir sous-optimale à un autre instant de la même journée. Une optimisation du routage des flux basée uniquement sur une matrice de trafic moyenne estimée en heure de pointe peut devenir moins pertinente.

L'optimisation et la reconfiguration dynamique du routage dans les réseaux constitue une alternative intéressante à une optimisation hors ligne reposant sur l'étude d'un pire cas. Ce schéma d'optimisation dynamique constitue l'essence de cette thèse : les travaux conduits visent la conception de méthodes d'optimisation dynamiques du routage, adaptant en temps réel les routes utilisées par les flux aux conditions de trafic dans le réseau. Nous considérons dans cette thèse deux points de vue :

1. le point de vue d'un opérateur réseau, qui est en capacité d'intervenir sur les équipements réseaux pour optimiser leur configuration.
2. le point de vue d'un fournisseur de service exploitant le réseau Internet existant, et ne pouvant pas intervenir directement sur sa configuration.

1.1 Plan de la thèse

Nous commençons par présenter dans le chapitre 2 quelques rappels sur les réseaux de communications IP, nécessaires à la compréhension des travaux présentés dans cette thèse. Nous y abordons les principaux protocoles régissant les communications sur le réseau internet et leurs spécificités : protocoles de routage, de transports. Nous y abordons aussi quelques notions primordiales sur la qualité de service dans les réseaux.

Le chapitre 3 est dédié à la présentation de quelques techniques de résolution de problème d'optimisation, directement applicables au domaine de l'optimisation des ressources réseau. Nous utiliserons certaines d'entre-elles dans les chapitres suivants.

Le chapitre 4 présente notre contribution concernant l'optimisation dynamique des poids OSPF pour le routage intra-domaine dans les réseaux IP, où le trafic est routé le long de plus courts chemins, en fonction des poids des liens. Nous proposons une approche en ligne permettant de reconfigurer dynamiquement les poids OSPF, et donc les routes utilisées, pour répondre aux variations observées du trafic et réduire ainsi le taux de congestion du réseau. L'approche proposée repose sur l'estimation robuste des demandes en trafic des flux à partir de mesures SNMP sur la charge des liens.

Les résultats expérimentaux, aussi bien sur des trafics simulés que réels, montrent que le taux de congestion du réseau peut être significativement réduit par rapport à une configuration statique.

Le chapitre 5 est dédié à l'optimisation du placement des LSP dans un réseau MPLS. Nous y proposons et étudions un algorithme inspiré de la théorie des jeux pour déterminer le placement des LSP optimisant un critère de performance non-linéaire. L'intérêt principal de l'algorithme proposé repose sur ses temps de calculs très réduits par rapport à d'autres méthodes, tout en prodiguant de bonnes solutions. Dans une première approche nous considérons un placement statique des LSP et étudions les performances de l'algorithme. Puis nous observons le comportement de l'algorithme dans le cadre d'une reconfiguration dynamique déclenchée périodiquement.

Dans le chapitre 6, nous considérons le point de vue d'un fournisseur de service. Nous concevons et développons une solution logicielle dédiée au déploiement d'un réseau overlay auto-guérissant et auto-optimisant entre plateformes de cloud. Ce dernier est conçu pour tenter de limiter l'impact des éventuelles défaillances (pannes ou mauvaises performances) du routage internet, en mesurant régulièrement la qualité des liens internet. Si un problème est détecté ou qu'un meilleur routage existe, le réseau overlay se charge de faire basculer le trafic sur les meilleures routes disponibles. La solution que nous proposons ne nécessite aucune modification des applications instrumentées. Nous décrivons dans ce chapitre l'architecture et la première implémentation du système, ainsi que des expériences menées pour valider son fonctionnement, réalisées à la fois en émulation et sur une plateforme réelle composée de plusieurs centres de données situés dans divers pays.

Les résultats du chapitre 4 ont été présentés à la conférence ITC25 [115] et publiés dans la revue Computer Networks [116]. Le travail exposé dans le chapitre 5 a été accepté pour une publication à la conférence INOC2015. Les travaux menés dans le chapitre 6 n'ont pas encore fait l'objet d'une publication.

2

Quelques éléments sur les réseaux de communications IP

2.1 Introduction

Dans ce chapitre, nous décrivons les grands principes relatifs au fonctionnement du réseau Internet, qui repose principalement sur la famille de protocoles TCP/IP. Afin de clarifier les hiérarchies et relations éventuelles entre protocoles, nous commencerons par aborder les deux principaux modèles définissant les couches de protocoles. Ces modèles permettent de classifier les protocoles suivant leur fonction et leurs interactions.

Nous aborderons ensuite le protocole Internet Protocol (IP), et la notion de routage, qui est au cœur des travaux présentés dans cette thèse. Nous continuerons ensuite en présentant les protocoles TCP et UDP, qui sont les deux principaux protocoles de transports d'informations utilisés aujourd'hui sur Internet.

Internet étant devenu une architecture de communication globale, supportant une grande diversité d'applications, il est important de connaître certaines notions pour exprimer la qualité de service dans les réseaux. Nous présenterons à cet effet quelques grandes métriques affectant l'expérience utilisateur : latence, gigue, pertes et bande passante. Enfin, nous terminerons ce chapitre en exposant quelques méthodes d'estimation de bande passante, qui nous serons utiles dans le dernier chapitre de cette thèse.

2.2 Modélisation standard des couches de protocoles

Les protocoles de communications utilisés peuvent être groupés suivant leur fonctionnalités et leur niveau de fonctionnement. On utilise pour cela des modèles représentatifs des différentes couches réseau. L'indépendance des couches y est assurée : un élément appartenant à une couche donnée ne doit pas intervenir sur une autre couche, et ne peut communiquer qu'avec un élément de même niveau. Les couches sont traversées successivement. Lorsqu'un message est émis par une application, les données sont transmises de la couche la plus haute vers la couche la plus basse, en traversant l'intégralité des couches intermédiaires.

On peut faire référence à deux grands modèles pour catégoriser un protocole réseau : le modèle Open Systems Interconnection (OSI) [124] et le modèle TCP/IP [23]. Une représentation graphique des deux modèles et de leurs équivalences est donnée en Figure 2.1.

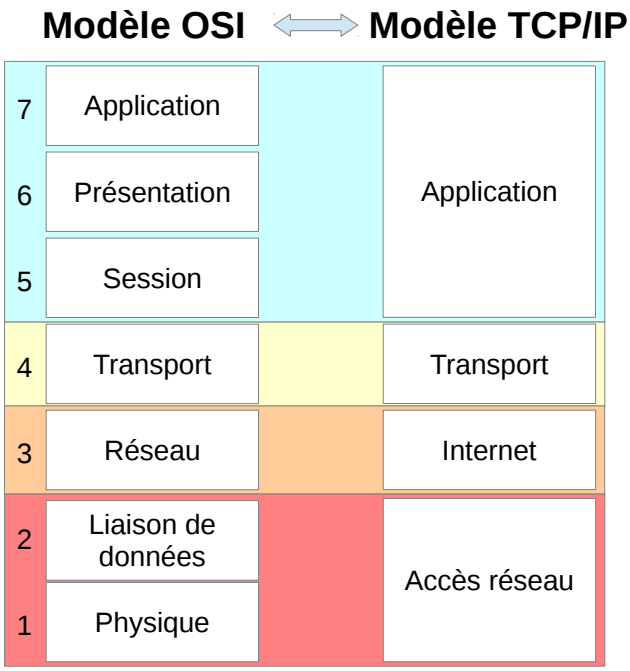


FIGURE 2.1 – Modèles OSI et TCP/IP.

Le modèle OSI est un modèle purement théorique normalisé par l'Organisation internationale de normalisation (ISO) en 1977. Il prend en compte l'hétérogénéité des équipements et définit comment ces derniers devraient communiquer.

Le modèle TCP/IP est calqué sur l'implémentation réelle des protocoles TCP et IP (que nous présenterons dans les sections 2.3 et 2.5.2). Ce modèle est maintenu par

l'organisation de standardisation nommée Internet Engineering Task Force (IETF). Il s'est progressivement imposé comme modèle de référence en lieu et place du modèle OSI. Il agrège certaines couches du modèle OSI (voir Figure 2.1), pour n'en conserver que 4 :

1. **Accès réseau** : regroupe les couches 1 et 2 du modèle OSI. Elle permet de transmettre un paquet IP sur le réseau.
2. **Internet** : cette couche gère l'acheminement des paquets dans les réseaux (potentiellement hétérogènes). Elle repose sur le protocole IP. Elle correspond à la couche 3 du modèle OSI. Nous évoquerons plus en détails le protocole IP dans la section 2.3.
3. **Transport** : correspondant à la couche 4 du modèle OSI, elle permet d'assurer la communication des données entre deux entités. TCP et UDP sont les deux principaux protocoles que l'on retrouve sur cette couche. Ces deux protocoles seront détaillés dans la section 2.5.
4. **Application** : regroupe les couches 5, 6 et 7 du modèle OSI. Ce regroupement s'explique par le fait que les couches 5 et 6 du modèle OSI sont en pratique très peu différenciées de la couche 7.

2.3 Protocole IP et routage

IP est un protocole permettant l'adressage et le routage des paquets. L'adresse IP permet d'identifier de manière unique une interface dans une machine connectée sur un réseau. Les paquets IP sont composés d'une entête, comportant principalement les adresses IP source et destination, et d'une zone réservée au transport des données de l'utilisateur.

Certaines machines, que l'on nomme routeurs, sont dotées de plusieurs interfaces réseaux, et sont chargées de transmettre et de router les paquets qu'elles reçoivent. Ces composants sont des points d'aiguillage qui forment l'architecture du réseau. À cet effet, chaque routeur dispose d'une table de routage lui permettant de définir sur quelle interface il devra transmettre le paquet qu'il a reçu, pour qu'il atteigne sa destination. Chaque paquet est alors routé de proche en proche, jusqu'à atteindre sa destination. Les échanges d'informations entre routeurs sont codifiées par les protocoles de routage qu'utilise le réseau. Ces échanges permettent aux routeurs de construire une image du réseau, et donc une table de routage.

Les réseaux IP composant l'Internet sont composés de groupes distincts de routeurs, que l'on nomme Système Autonome (AS). Chaque AS regroupe des routeurs placés sous la même autorité administrative, et est géré indépendamment. Le nombre d'AS ne cesse de croître, et en début 2015, le nombre d'AS présents sur Internet est d'environ 50000 [112].

La construction de la table de routage utilisée par le routeur pour transmettre les paquets s'effectue différemment à l'intérieur d'un AS (on parle de routage intra-domaine) et entre les différents AS (routage inter-domaine). La complémentarité des deux types de routage est représentée sur la Figure 2.2.

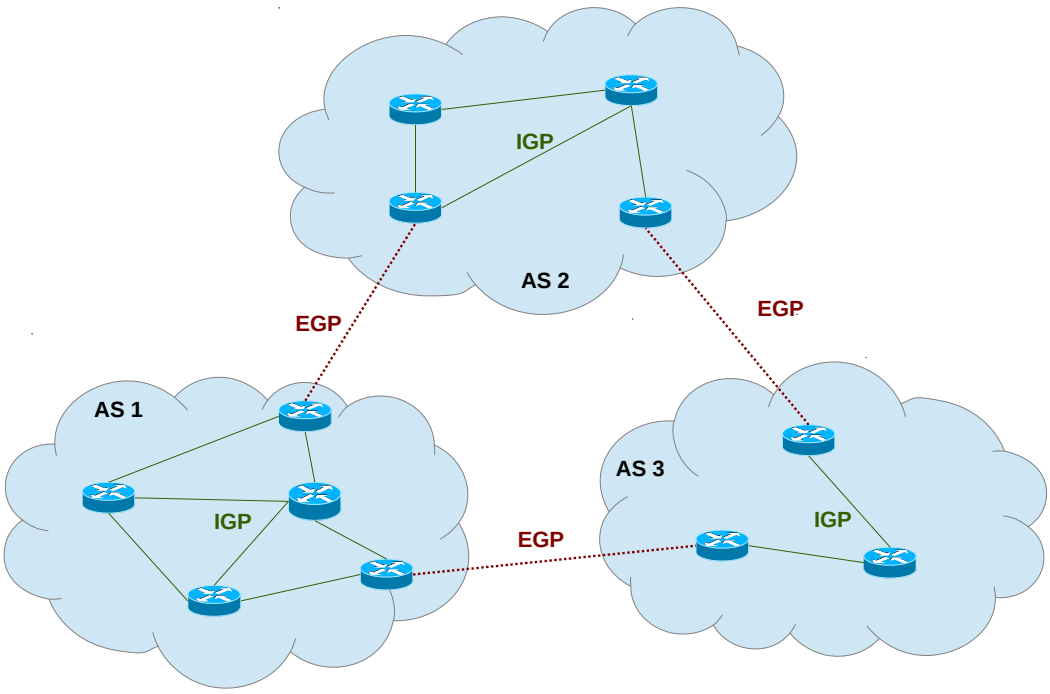


FIGURE 2.2 – Routage intra-domaine avec IGP et inter-domaine avec EGP.

Nous évoquons dans les sections suivantes les spécificités des principaux protocoles utilisés sur Internet pour ces deux types de routage : OSPF pour le routage intra-domaine et BGP pour le routage inter-domaine.

2.3.1 Routage intra-domaine avec IGP

Le routage à l'intérieur même d'un AS, aussi appelé routage intra-domaine, s'effectue à l'aide de protocoles dédiés, que l'on nomme Interior Gateway Protocol (IGP). On peut alors citer les protocoles Open Shortest Path First (OSPF) [78], Intermediate System to Intermediate System (IS-IS) [86] et dans un moindre mesure Routing Information Protocol (RIP) [71]. Nous présentons ici le protocole OSPF, qui est le plus répandu et avec lequel nous travaillerons dans le chapitre 4. Son fonctionnement est très similaire à celui du protocole IS-IS.

OSPF est un protocole de routage à état de lien, qui fut originellement crée par

l'IETF pour remplacer le protocole RIP. Les routeurs utilisant OSPF s'échangent l'état des liens qui leur sont rattachés. Cette communication permet à chaque routeur de construire une image du réseau, qui n'est mise à jour qu'en cas de modification de topologie (comme la suppression et l'ajout de liens ou de nœuds). OSPF procède malgré tout à un envoi périodique de l'intégralité des informations d'états des liens (toutes les 30 minutes par défaut).

Comme le nom du protocole l'indique, les paquets sont routés sur des plus courts chemins. La table de routage de chaque routeur est donc déterminée à partir d'un algorithme de plus courts chemins (e.g. algorithme de Dijkstra [37]). La longueur d'un chemin est définie comme la somme des longueurs des liens parcourus, tandis que la longueur d'un lien (que l'on nomme également métrique ou poids OSPF) peut-être fixée arbitrairement par le gestionnaire du réseau (à la seule condition d'être un nombre entier positif). Les poids OSPF sont souvent fixés à une valeur égale à 1 (la longueur d'un chemin représente alors le nombre de sauts), mais certains opérateurs comme CISCO, proposent d'utiliser un poids tenant compte de la capacité du lien (poids = 10^8 / capacité). Ces poids influencent la longueur des chemins, donc les plus courts chemins et le routage qui en résulte. Lorsque plusieurs routes de même longueur sont disponibles, un partage de charge équitable entre les chemins est employé : s'il existe n chemins d'égales longueurs reliant le routeur courant s à une destination t, alors chaque flux passant par s à destination de t avec une demande d sera routé sur les n chemins, chacun faisant alors transiter une demande égale à d/n. Nous exploiterons ces propriétés dans le chapitre 4 consacré à l'optimisation du routage OSPF.

2.3.2 Routage inter-domaine avec EGP

Le routage entre AS est assuré par un Exterior Gateway Protocol (EGP), dont le plus communément utilisé est Border Gateway Protocol (BGP). Il s'agit d'un protocole dont la scalabilité est reconnue, BGP étant en effet responsable de la propagation des routes Internet à l'échelle mondiale. Il ne tient pas compte de la structure interne des AS (cette partie étant dévolue aux IGP).

Les voisins BGP échangent toutes leurs informations dès leur première connexion en utilisant le protocole TCP (qui sera présenté en section 2.5.2). Ensuite, seules les éventuelles informations de mise à jour sont transmises, si une route est modifiée ou ajoutée. Contrairement à OSPF, le protocole BGP ne procède pas à un envoi périodique des informations. La propagation des routes de bout en bout s'effectue plus exactement grâce à deux protocoles distincts : BGP externe (e-BGP) pour la communication de routes directement entre routeurs de frontière d'AS différents, et BGP interne (i-BGP) pour l'échange de routes passant à l'intérieur des AS. Nous illustrons la distinction entre i-BGP et e-BGP sur la Figure 2.3.

Malgré sa grande scalabilité, l'utilisation de ce protocole entraîne parfois certains

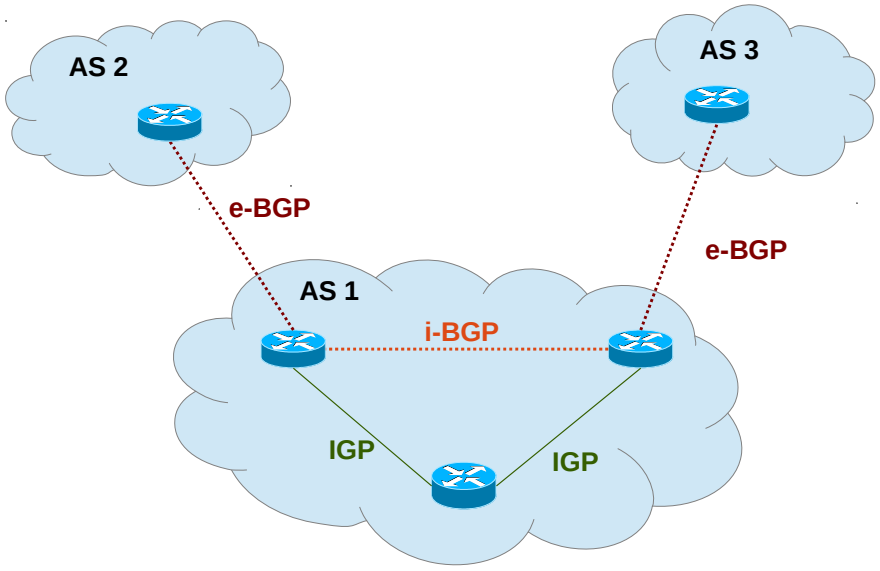


FIGURE 2.3 – Protocole BGP : distinction entre i-BGP et e-BGP.

problèmes d'optimalité et de résilience des routes Internet. Nous reviendrons sur les différents problèmes qui peuvent être constatés lors de l'utilisation de ce protocole dans le chapitre 6. Nous y proposerons une solution à base de réseau overlay pour améliorer le routage internet.

2.4 La technologie MPLS

La technologie Multi Protocol Label Switching (MPLS) est une technologie réseau particulière qui a été mise au point et proposée pour répondre à deux principaux objectifs : premièrement, permettre un acheminement rapide des paquets en limitant le plus possible le temps passé dans les routeurs. Deuxièmement, prodiguer les outils nécessaires aux opérateurs pour maîtriser plus facilement l'acheminement des données dans leurs cœurs de réseau.

MPLS est un protocole se situant entre les couches 2 et 3 du modèle OSI. Pour répondre aux objectifs fixés, MPLS remplace la fonction de routage des routeurs par une fonction de commutation d'étiquette, bien plus efficace. L'idée principale est d'associer et d'ajouter une étiquette aux paquets IP entrant sur le réseau, au niveau du routeur d'entrée nommé Label Edge Router (LER). Cette étiquette est ensuite utilisée par tous les routeurs intermédiaires, que l'on nomme Label Switch Router (LSR), afin d'acheminer les paquets. Les LSR tiennent uniquement compte des labels pour router les paquets (et peuvent donc faire abstraction complète des adresses IP). Lorsque le paquet arrive à son LER destination (pour sortir du réseau MPLS) l'en-tête MPLS est supprimé du

paquet, qui peut dès lors continuer son chemin hors du réseau. MPLS utilise des Label Switching Path (LSP) pour agréger et transporter les trafics. Les LSP encapsulent les paquets et déterminent précisément la route empruntée, via l'utilisation des étiquettes. MPLS permet aussi de créer des classes de trafic nommées Forward Equivalent Class (FEC), qui déterminent des classes de service. La combinaison de l'utilisation des LSP et des FEC rend MPLS particulièrement adapté à la différenciation des classes de service dans le réseau : il suffit d'affecter une étiquette à un paquet en fonction de sa FEC pour opérer un routage dépendant de la classe de trafic.

De façon plus générale, les labels peuvent être associés aux LSP en fonction de multiples critères tels que la source, la destination, un chemin particulier, une FEC, etc... Ce fonctionnement permet d'étendre le routage IP classique.

La possibilité offerte par MPLS de déterminer un chemin spécifique par LSP permet aux opérateurs d'adapter la gestion du trafic à leurs besoins. Nous étudierons le problème de l'optimisation du placement des LSP dans le chapitre 5 de cette thèse.

2.5 Protocoles de la couche transport

Nous présentons ici les deux principaux protocoles utilisés sur la couche transport (4ème niveau de la couche OSI) : UDP et TCP. Ces deux protocoles s'utilisent de manière complémentaire.

2.5.1 UDP

User Datagram Protocol (UDP) [95] est un protocole léger fournissant les mécanismes primaires permettant à une application d'envoyer des données à d'autres applications sur le réseau. Il propose la notion de port pour différencier les différentes applications utilisant UDP sur une même machine. Un message UDP est associé à un port sur la machine source et à un port sur la machine destination. Lors de la réception, les paquets sont automatiquement redirigés par le système d'exploitation vers la bonne application en fonction du port destination utilisé.

UDP est encapsulé dans les paquets IP, ce dernier se chargeant de la transmission effective des données sur le réseau. Mais tout comme IP, UDP ne fournit aucun mécanisme supplémentaire pour assurer la fiabilité de la transmission. Les transmissions s'effectuent alors dans un mode déconnecté et non fiable. Ainsi, les messages UDP peuvent être perdus, dupliqués, ou même arriver à destination de façon désordonnée. De plus, si la source envoie des paquets trop rapidement pour que l'application réceptrice puisse tous les traiter, des pertes de paquets seront observées. L'intégrité des données transportées par un paquet est la seule garantie apportée par le protocole via l'utilisation d'un checksum intégré dans l'en-tête UDP (tout comme IP).

L'absence de mécanisme évolué de contrôle permet à UDP de proposer la transmission la plus légère possible. Une application utilisant le protocole UDP souhaitant une gestion plus poussée des paquets devra nécessairement intégrer ce traitement à l'intérieur même de l'application. Les applications de type streaming, qui intègrent le contrôle d'erreur dans leurs algorithmes, sont de très bons exemples de l'utilisation d'UDP. Les applications ne souhaitant pas gérer par elle-même le contrôle des paquets peuvent se tourner vers le protocole TCP, que nous abordons dans la section suivante.

2.5.2 TCP

De la même manière qu'UDP, le protocole Transmission Control Protocol (TCP) [96] prodigue aux applications le moyen de communiquer entre elles sur le réseau. Il repose aussi sur le protocole IP pour transférer ses paquets sur le réseau, et introduit également la notion de port pour différencier les différentes destinations possibles s'exécutant sur une même machine. Mais contrairement à UDP, le protocole TCP est un protocole de transport complexe, fiable, fonctionnant en mode connecté. Nous n'abordons ici que les grandes lignes de son fonctionnement.

La procédure de connexion suivie par TCP est composée de trois étapes connues sous le nom de "Three-way handshake". La source commence par envoyer une demande de connexion vers la destination. Si la destination accepte la connexion, elle envoie un message vers la source, qui lui répond enfin pour confirmer l'ouverture de la connexion. Une fois la connexion établie, les applications peuvent communiquer. Le protocole utilise des messages d'acquittement (ACK) (envoyé par le récepteur lors de la réception d'un paquet) pour détecter les erreurs de transmissions. Si le message d'acquittement correspondant à un message transmis précédemment n'est pas reçu après une certaine période de temps, le message est ré-émis. De la même manière à la réception, TCP procède à un contrôle des paquets : les messages dupliqués sont ignorés et les messages désordonnés sont réordonnés avant leur transfert à l'application destination.

Le protocole TCP adapte automatiquement le débit de transfert en procédant à un contrôle de congestion. Il utilise pour cela une fenêtre glissante de taille variable définissant le nombre maximum de paquets pouvant transiter sur le réseau avant qu'un paquet d'acquittement ne soit envoyé vers la source. La taille de la fenêtre varie différemment suivant deux phases successives : phase Slow Start(SS) puis phase Congestion Avoidance (CA). Nous présentons la succession des deux phases sur la Figure 2.4.

La phase SS correspond à une augmentation exponentielle du débit d'émission, jusqu'à ce qu'un certain seuil soit atteint, ou qu'une perte de paquet soit constatée. Si un paquet est perdu, la phase redémarre depuis le début. Par contre si aucune perte de paquet n'est constatée, l'algorithme passe alors dans la phase CA. La taille de la fenêtre est alors incrémentée progressivement, jusqu'à ce qu'une perte de paquet soit détectée. Dès qu'une perte est constatée, la taille de la fenêtre glissante est divisée par 2, puis

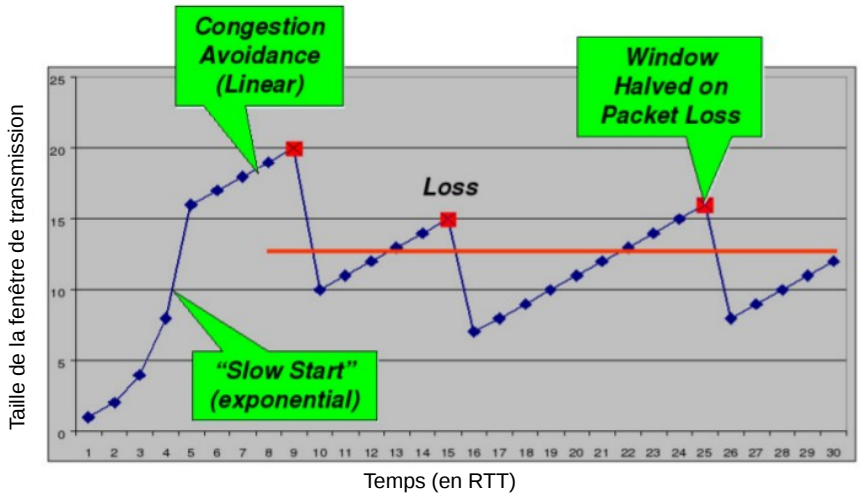


FIGURE 2.4 – Évolution de la taille de la fenêtre TCP.

l'algorithme recommence son augmentation progressive.

2.6 Métriques de qualité de service dans les réseaux

Les applications utilisant le réseau peuvent être de toute nature et fournir toutes sortes de services aux utilisateurs. Cette multitude de services rendus implique souvent des besoins différents vis à vis du réseau sous-jacent. Un transfert de fichier nécessitera par exemple une connexion fiable et un transfert le plus rapide possible (mais le transfert d'un petit fichier sera lui très sensible à la latence). Les applications de streaming, qui se développent de plus en plus aujourd'hui, sont moins sensibles aux pertes car elles intègrent des dispositifs de contrôle d'erreur, mais sont très sensibles à la latence et aux variations de cette dernière (gigue). Nous rappelons dans cette section quelques métriques susceptibles d'intéresser l'utilisateur final vis à vis de son utilisation des services réseau. Nous aborderons la latence, puis le taux de perte, et enfin la bande passante disponible.

2.6.1 Latence

La latence dans les réseaux se définit très simplement : il s'agit du temps total nécessaire à la transmission d'un paquet entre sa source et sa destination. On parle du Round Trip Time (RTT) pour qualifier le temps nécessaire à un aller-retour. Cette latence est tout d'abord occasionnée par un phénomène physique : les paquets sont transmis sous la forme d'un signal qui ne peut dépasser la vitesse de la lumière (300000 km/s dans le vide). Ainsi, si l'on considère une transmission entre les villes de Paris et de Los Angeles (distantes d'environ 9000 km), la latence de transmission sera nécessairement supérieure à 30 ms. Les communications par satellites sont particulièrement touchées par ce phénomène, en raison de la grande distance qui les sépare du sol (35000 km environ en position géostationnaire).

Mais d'autres contraintes s'ajoutent à ce délai minimum de transmission, car les paquets doivent transiter par de nombreux équipements pour être acheminés jusqu'à leur destination. Chaque équipement ne dispose que d'une capacité de traitement finie qui détermine le nombre maximum de paquets pouvant être traités par unité de temps. Les paquets doivent patienter dans les files d'attente de chaque équipement avant d'être traités.

Les conséquences de la latence sont particulièrement visibles pour tous les services qui fonctionnent en temps réel : transmission vidéo, communication vocale, applications interactives de type jeux-vidéo, etc. La gigue, qui représente les variations de latence au cours du temps, a également une forte influence sur ces applications.

2.6.2 Taux de perte

Le taux de perte correspond au nombre de paquets qui n'arrivent pas correctement jusqu'à leurs destinations. Ce phénomène peut être principalement causé par deux fac-

teurs. Le premier réside dans un dépassement des capacités intrinsèques de traitement du réseau de transmission : si les équipements traversés deviennent trop congestionnés (remplissant leur files d'attente plus rapidement qu'ils ne peuvent les vider) des paquets sont perdus, faute de pouvoir être stockés en file d'attente.

Un deuxième facteur réside dans la corruption éventuelle des paquets lors de leur transmission. Les signaux transportant les paquets peuvent en effet subir des influences externes (interférences par exemple) causant une corruption du signal transmis. Les connexions sans fil (wi-fi, satellite, etc...) sont particulièrement concernées. La somme de contrôle employée par les protocoles de la pile TCP/IP est conçue pour détecter ce type de corruption. Lorsque l'équipement traversé calcule une somme de contrôle différente de celle attendue, le paquet est rejeté.

Comme pour le délai, le taux de perte influence négativement la qualité de service des applications temps réel comme la communication audio ou vidéo. Bien qu'ils intègrent ce problème dans leur conception, des taux de pertes élevées dégradent la qualité perçue par l'utilisateur (dégradation de la qualité vidéo ou audio, coupures intempestives). Les connexions TCP, qui sont immunisées contre les pertes de paquets, subissent indirectement leur influence : le débit maximal atteignable avec TCP est en effet fortement influencé par ce paramètre, comme nous le verrons dans la section qui suit.

2.6.3 Bande passante disponible

Cette notion reflète la capacité d'un chemin à accueillir du trafic supplémentaire, et donc à faire face à une augmentation de trafic. Le terme de "bande passante disponible" porte parfois à confusion, car il peut se rapporter à différents types de notions. Nous effectuons ici une brève description des notions souvent rattachées (parfois abusivement) au terme de bande passante disponible sur un lien.

— **Capacité :** il s'agit de la notion dont dépendent nécessairement toutes les autres. Elle indique la quantité maximale de données pouvant transiter sur le lien par unité de temps. Il s'agit d'une propriété du lien, qui dépend aussi bien de ses propriétés physiques que de sa configuration. Le gestionnaire du lien peut délibérément décider de limiter cette capacité en jouant sur la configuration logicielle. Cette capacité est considérée comme fixe tant qu'aucune modification n'est opérée sur le lien.

— **Bande passante disponible :** elle se rapporte à la capacité inutilisée d'un lien par unité de temps. Elle dépend de la capacité du lien mais aussi de la quantité de données qui y circule effectivement. Cette métrique est une variable dépendante du temps. Si l'on note $A_i(t)$ la bande passante disponible à l'instant t, C_i la capacité du lien i, et $u_i(t)$ la quantité de données transitant sur le lien i, alors :

$$A_i(t) = C_i - u_i(t) \tag{2.1}$$

— **Débit effectif TCP :** décrit la quantité de trafic maximale pouvant être transportée par un flux TCP sur le lien considéré à l'instant de mesure. Cette notion est dépendante d'un nombre important de facteurs : elle dépend ainsi du type de trafic transitant déjà sur le lien considéré (UDP, TCP...), du nombre de connexions TCP, de la taille des buffers TCP, de la congestion le long du chemin de retour pour l'envoi des paquets ACK, de l'algorithme utilisé pour l'implémentation de TCP, etc. Cette notion est différente de la bande passante totale disponible : ainsi sur un lien de capacité C_i saturé par une unique connexion TCP, la bande passante disponible serait égale à 0, mais le débit effectif TCP peut potentiellement valoir $C_i/2$ en raison du partage équitable de bande passante opéré par TCP.

Il est bon de noter qu'il faut différencier la bande passante d'un lien de la bande passante d'un chemin. Ainsi la bande passante d'un chemin est égale à la bande passante minimale de tous les liens constituant le chemin, et donc de son "goulot d'étranglement". Sur un chemin π la bande passante disponible vaut :

$$A^{\pi}(t) = \min_{i \in \pi} A_i(t) \tag{2.2}$$

2.7 Quelques techniques de mesure de bande passante utilisables coté client

Nous considérons ici les méthodes de mesure de bande passante qu'un client peut utiliser sans connaissance des spécificités du réseau traversé. Cette mesure étant dépendante du trafic, elle se doit d'être la moins intrusive possible. Nous commençons par présenter une méthode d'estimation passive dédiée au protocole TCP. Nous présentons ensuite deux techniques de mesures actives. Nous évaluerons certaines de ces techniques dans le chapitre 6 de ce mémoire.

2.7.1 Techniques d'estimation passives pour le protocole TCP

Nous présentons ici une méthode permettant d'approximer la bande passante maximale pouvant être atteinte avec le protocole TCP, reposant uniquement sur les mesures de RTT et de perte de paquets. Il s'agit d'un résultat basé sur un modèle très simplifié de TCP [73] dans la phase CA (Congestion Avoidance, cf présentation de TCP dans la section 2.5.2). Ce modèle simplifié est présenté dans la Figure 2.5.

Dans ce modèle, on considère uniquement la phase CA. Cette hypothèse reste valide pour les connexions persistantes, pour lesquelles la majorité du temps sera passé dans cette phase. Lorsqu'une perte de paquet est constatée, la taille de la fenêtre W(t) est divisée par 2. Dans ce modèle, la taille de la fenêtre W(t) évolue entre W/2 et W, où W est la taille maximale de la fenêtre.

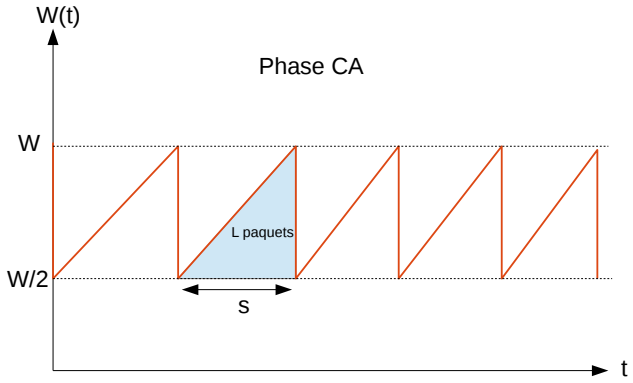


FIGURE 2.5 – Modèle simplifié de la phase CA concernant l'évolution de la taille de la fenêtre TCP.

Si l'on note L le nombre moyen de paquets transmis dans un cycle, et S la durée moyenne du cycle, alors le débit moyen vaut $D = L/S$. La taille de la fenêtre augmentant de 1 à chaque RTT, la durée moyenne du cycle est alors :

$$S = RTT\frac{W}{2} \tag{2.3}$$

Le nombre moyen de paquets transmis durant un cycle est alors :

$$L = \sum_{i=0}^{W/2}(\frac{W}{2} + i) = \frac{3W^2 + 5}{8} \approx \frac{3W^2}{8} \quad \text{paquets / seconde} \tag{2.4}$$

Dans une première approche, on peut choisir de négliger les pertes de paquets. Dans ce cas, à partir des équations 2.3 et 2.4, le débit moyen maximal atteignable dans la phase CA s'exprime ainsi :

$$D = \frac{L}{S} = \frac{3W}{4RTT} \quad \text{paquets / seconde} \tag{2.5}$$

Si l'on considère que chaque paquet peut être perdu avec une probabilité p, on a une distribution géométrique du nombre de paquets transmis avant perte, $p_i = (1-p)^{i-1}p$ la probabilité de transmettre i paquets avant d'avoir une perte. Le nombre moyen de paquets transmis avant une perte est alors :

$$L = \sum_i ip_i = \frac{1}{p} \tag{2.6}$$

En tenant compte de l'équation 2.4, on trouve $L = \frac{3W^2}{8}$. On en déduit immédiatement une formulation du débit moyen TCP en fonction du taux de perte, connue sous

le nom de Square Root Throughput (SQRT) [73] :

$$D = \frac{3W}{4RTT} = \frac{1}{RTT}\sqrt{\frac{3}{2p}} \quad \text{paquets / seconde} \tag{2.7}$$

2.7.2 Techniques de mesure actives

Nous évoquons ici les techniques de métrologie actives, pour lesquelles des paquets de mesure sont spécifiquement introduits dans le réseau.

Une approche naïve pour mesurer la bande passante consisterait à transférer avec TCP un grand nombre de paquets en mesurant le temps nécessaire pour le transfert. Cette approche implique un envoi massif de données et se doit donc d'être évitée autant que possible. Des méthodes beaucoup plus fines ont été développées dans la littérature pour obtenir des estimations de la bande passante disponible.

La technique Self-Loading Periodic Streams (SLoPS) est une méthode de mesure de bande passante disponible sur un chemin qui fut introduite dans [64]. Avec cette méthode, la source envoie en UDP un ensemble de groupes de paquets (composé d'une centaine de paquets de taille identique) jusqu'au récepteur, avec différents débits. La méthode consiste à observer les variations du délai de transmission de ce groupe de paquets en fonction du débit d'envoi D. Les échanges entre source et destination sont illustrés sur la Figure 2.6, où A est la source et B la destination. Le nœud A envoie N paquets de taille L aux instants t_i^A, $1 \leq i \leq N$ en direction du nœud B, avec un débit D. Le nœud B note la date d'arrivée de chaque paquet t_i^B. Dès que le nœud B a reçu l'intégralité des N paquets (ou qu'un timeout est écoulé en cas de perte de paquet), il communique alors à la source A un paquet contenant les délais $(t_i^B - t_i^A)$ mesurés.

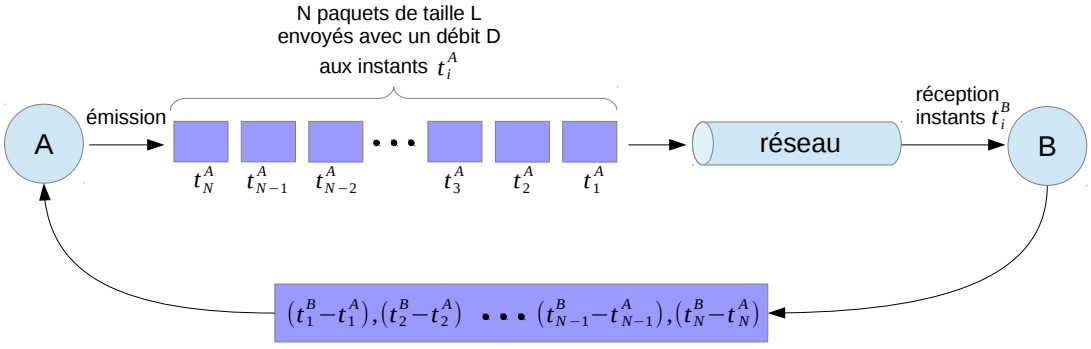


FIGURE 2.6 – Méthode SLoPS : mesure des délais.

On peut remarquer que cette façon de mesurer les délais constitue l'une des contraintes de la méthode : pour fonctionner correctement, il faut que les horloges de A et B soient parfaitement synchronisées, ce qui est difficile dans la pratique.

En fonction des délais reçus, la source décide de faire varier le débit D. Tant que le débit D ne dépasse pas la bande passante disponible A, on ne constate pas de variation notable des délais associés à chaque paquet. Mais, dès que D devient supérieur à A, le flux de paquets est temporisé pendant un certain temps au niveau du goulot d'étranglement. Le temps de temporisation augmente alors à mesure que le débit D augmente. Nous illustrons ce phénomène sur la Figure 2.7, où les courbes des délais mesurés sont représentés pour deux débits d'envoi distincts ($D1 < A$ et $D2 > A$).

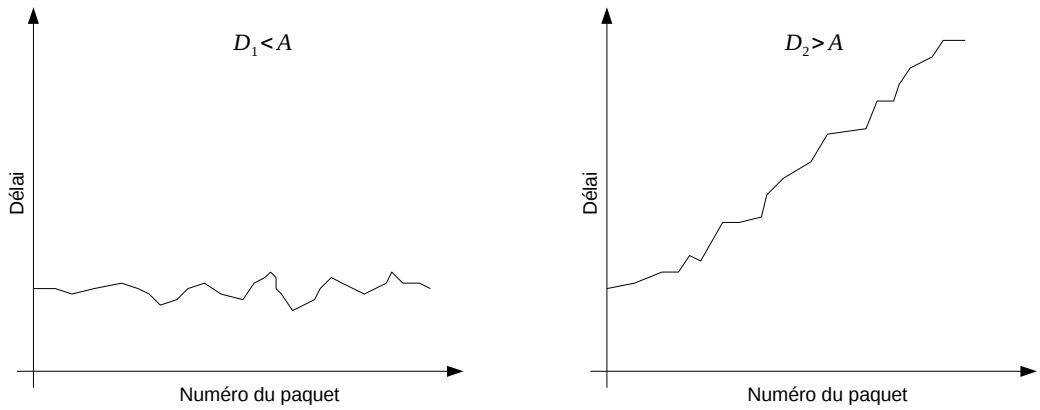


FIGURE 2.7 – Méthode SLoPS : augmentation des délais mesurés si l'on dépasse la bande passante disponible.

L'idée de la méthode consiste à trouver le point d'inflexion A à partir duquel on constate une augmentation des délais des paquets. Les auteurs de la méthode proposent pour cela de procéder sous la forme d'une recherche binaire. Pour s'assurer que les paquets de mesure ne perturbent pas trop le système, l'émetteur respecte une période de repos entre deux groupes d'envoi successifs.

Une autre méthode de mesure de bande passante, nommée Trains of packet pairs (TOPP), fut introduite dans [75] et [76]. Cette méthode repose sur les mêmes hypothèses de modélisation du goulot d'étranglement, mais la mise en œuvre de la mesure s'effectue différemment. Ici, nous n'envoyons plus de longues séquences de paquets : seuls 2 paquets de taille L doivent être émis à chaque itération de la méthode. Les paires de paquets sont envoyées successivement, avec un débit D croissant à chaque nouveau couple. Conformément au débit D considéré, l'écart temporel à l'émission Δ_s entre les deux paquets d'un couple respecte la formule $D = L/\Delta_s$. Lorsque D est supérieur à A, l'écart temporel entre les deux paquets au niveau de la réception Δ_t est impacté et devient supérieur à Δ_s.

Nous illustrons les transferts nécessaires entre nœuds source et destination sur la

Figure 2.8. Le phénomène typique d'augmentation du délai à partir d'un certain débit D est présenté sur la Figure 2.9. Pour TOPP, la recherche du point d'inflexion correspondant à la bande passante A s'effectue via une recherche linéaire, en commençant par de petits débits d'émission.

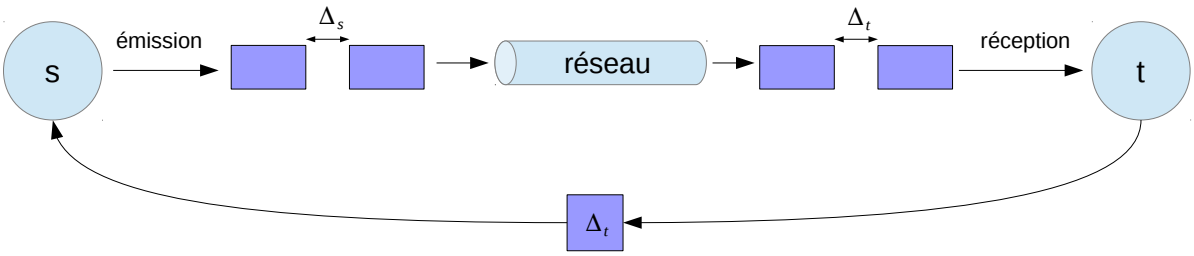


FIGURE 2.8 – Méthode TOPP : transferts entre source et destination.

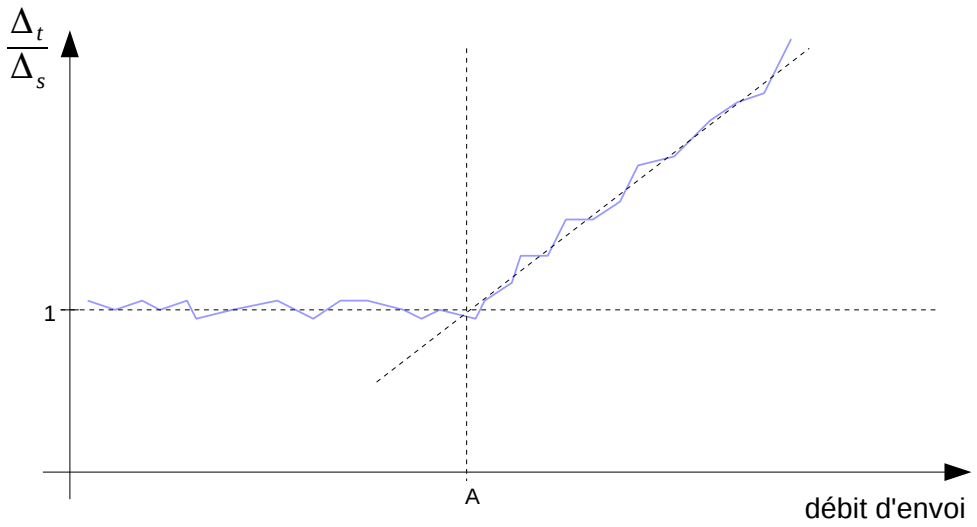


FIGURE 2.9 – Méthode TOPP : augmentation théorique du délai à la réception en fonction du débit d'envoi.

Les auteurs indiquent que l'une des difficultés de la méthode TOPP réside dans son comportement complexe dès lors que le nombre de liens traversé augmente, car plusieurs phénomènes de bufferisation peuvent alors apparaître sur les courbes. Un filtrage avancé des résultats obtenus est alors nécessaire. De plus, lorsque la capacité du lien devient très importante, l'utilisation de deux paquets successifs ne suffit plus pour observer correctement un écart temporel : il faut alors augmenter le nombre de paquets.
Nous reviendrons plus en détails sur la précision des résultats obtenus via ces méthodes dans le chapitre 6.

2.8 Conclusion

Dans ce chapitre, nous avons brièvement décrit le fonctionnement des réseaux de communication de type IP, et les différents protocoles ou notions nécessaires à la compréhension des travaux présentés dans cette thèse. De la même manière, avant d'exposer nos contributions, nous présentons dans le chapitre qui suit un certain nombre de techniques d'optimisation qu'il est important de connaître.

3

Techniques d'optimisation

3.1 Introduction

Les techniques d'optimisation utilisées pour déterminer un routage optimal font appel à la théorie de l'optimisation, que nous abordons dans ce chapitre. Cette théorie s'applique à de très nombreux problèmes : de nombreux sujets aussi bien pratiques que théoriques, nécessitent de trouver la ou les "meilleures" configurations permettant d'atteindre un objectif particulier. Les problèmes d'optimisation se regroupent en plusieurs classes, suivant leurs propriétés. Nous rappelons les principales catégories de problèmes d'optimisation sur la figure 3.1.

L'ensemble le plus général est celui des problèmes non-linéaires. Ils s'expriment sous la forme générale suivante :

$$\begin{aligned} &\text{minimiser } f(x) \\ &\quad \text{avec } g_i(x) \geq 0 \quad i = 1, \ldots, m && (3.1) \\ &\quad\quad h_j(x) = 0 \quad j = 1, \ldots, p && (3.2) \\ &\quad\quad x \in \mathbb{R}^n && (3.3) \end{aligned}$$

L'objectif est alors de trouver le point x respectant les conditions 3.1, 3.2 et 3.3, pour laquelle la valeur de la fonction $f(x)$ est minimale. Bien que cette formulation soit basée sur la recherche d'un minimum, elle s'applique également à la recherche d'un maximum : il suffit pour cela de remplacer la fonction $f(x)$ par $-f(x)$. Il s'agit de la formulation la

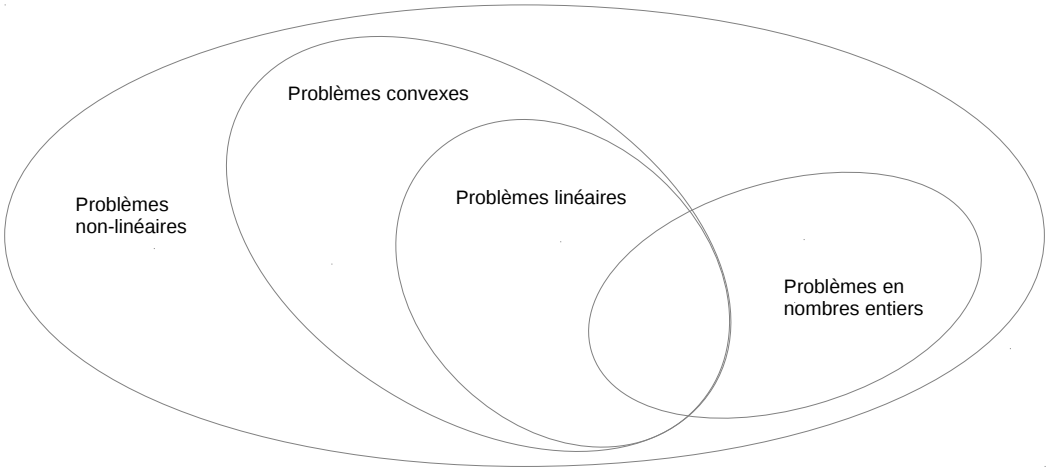


FIGURE 3.1 – Classification des problèmes d'optimisation

plus générale : les fonctions f, g_i et h_j y sont quelconques. Les caractéristiques associées à ces trois fonctions permettent de déterminer la classe à laquelle appartient le problème, et par voie de conséquence le type de méthodes qui devront être utilisées pour le résoudre.

Lorsque la fonction f est convexe, g_i concave ($\forall i$) et h_j linéaire ($\forall j$), on dit que le problème est convexe. Si toutes ces fonctions sont linéaires, on dit que le problème appartient à la classe des problèmes linéaires. Par ailleurs, il existe des problèmes où les variables sont de type entier : on parle de problèmes d'optimisation en nombres entiers. Ces derniers peuvent appartenir en même temps à la classe des problèmes non-linéaires, convexes, ou linéaires.

Il existe deux principaux types de méthodes d'optimisation : les méthodes de résolution exacte et les méthodes de résolution approchée, également appelées heuristiques. Nous présentons dans ce chapitre un panorama non exhaustif de techniques couramment utilisées dans le domaine de l'optimisation, appartenant aux deux types de familles.

3.2 Optimisation linéaire

La programmation linéaire est un outil très puissant permettant d'exprimer et de résoudre les problèmes du même nom. On peut distinguer la programmation linéaire en nombre réels, où les variables appartiennent à l'ensemble $\mathbb{R}$, de la programmation en nombre entiers où les variables appartiennent à l'ensemble $\mathbb{N}$. On appelle programmation linéaire mixte le cas où un problème requiert l'appartenance d'une partie de ses variables dans $\mathbb{R}$ et de l'autre partie dans $\mathbb{N}$.

3.2.1 Programmation linéaire

Les programmes linéaires appartiennent à la classe des problèmes convexes, l'ensemble de définition des solutions y appartient donc également. Il s'agit d'un ensemble fermé et fini de solutions, défini sous la forme d'un polyèdre S appartenant à $\mathbb{R}^n$. Il a été établi que toute solution optimale du problème d'optimisation est forcément un sommet du polyèdre [83].

L'algorithme du Simplexe utilise principalement cette propriété pour atteindre l'optimum. Il fut proposé en 1947 par George Dantzig, mais reste aujourd'hui encore l'un des plus utilisés pour la résolution de problèmes linéaires avec des variables continues. Cette méthode permet de trouver une solution en un nombre fini d'étapes. Bien que sa complexité théorique au pire cas soit de type exponentielle, l'algorithme parvient à atteindre en pratique une remarquable efficacité.

Le fonctionnement de l'algorithme consiste à parcourir les arêtes du polyèdre constituant l'ensemble des solutions, en passant d'une extrémité d'arête à l'autre. En un point donné, l'algorithme définit la prochaine arête traversée comme étant celle permettant de faire décroître la fonction coût le plus possible. Un exemple de trajectoire suivie par l'algorithme du simplexe est présentée sur la figure 3.2.

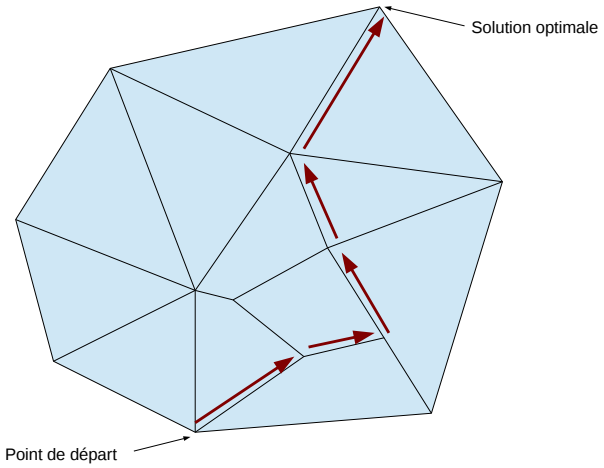


FIGURE 3.2 – Trajectoire suivie par l'algorithme du Simplexe sur le polyèdre des solutions

Malgré sa grande efficacité en moyenne, la complexité exponentielle de l'algorithme dans les pires cas a entraîné l'émergence des algorithmes de type points intérieurs [97], pour lesquels la complexité au pire cas est polynomiale. A l'inverse de l'algorithme du simplexe, ces méthodes suivent une trajectoire passant par l'intérieur du polyèdre des solutions, et utilisent des fonctions de type barrière pour orienter la recherche. En pratique, et malgré la complexité polynomiale au pire cas, elles ne fournissent pas nécessairement les solutions plus rapidement que les méthodes basées sur l'utilisation du Simplexe.

Les algorithmes actuellement implémentés dans les principales solutions logicielles de résolution de problèmes linéaires avec variables continues se basent principalement sur ces deux grandes méthodes.

3.2.2 Programmation linéaire en nombre entiers

Certains problèmes requièrent explicitement l'utilisation de nombres entiers. C'est le cas de tous les modèles représentant des instances qui ne peuvent être fractionnées (le routage d'un flux sur un chemin unique en est un exemple).

La résolution d'un problème linéaire en nombre entier ou mixte est singulièrement plus complexe que celle d'un problème linéaire en nombre réels, et les méthodes du Simplexe et des points intérieurs ne s'appliquent pas sur ce type de problème. Le problème NP-complet du voyageur de commerce appartient par exemple à cette catégorie [88].

Une approche naïve de ce type de problème consisterait à résoudre le problème sans les contraintes liant les solutions à l'espace des entiers, puis à considérer la solution entière la plus proche de la solution trouvée. Bien que séduisante au premier abord, cette méthode ne fournit malheureusement pas les bonnes solutions. Nous illustrons ainsi sur la figure 3.3 un exemple simple à deux variables où cette méthode ne fonctionne pas. Sur cet exemple, aucun des 4 points entiers les plus proches de la solution optimale avec variables continues n'est compris dans le polyèdre des contraintes S.

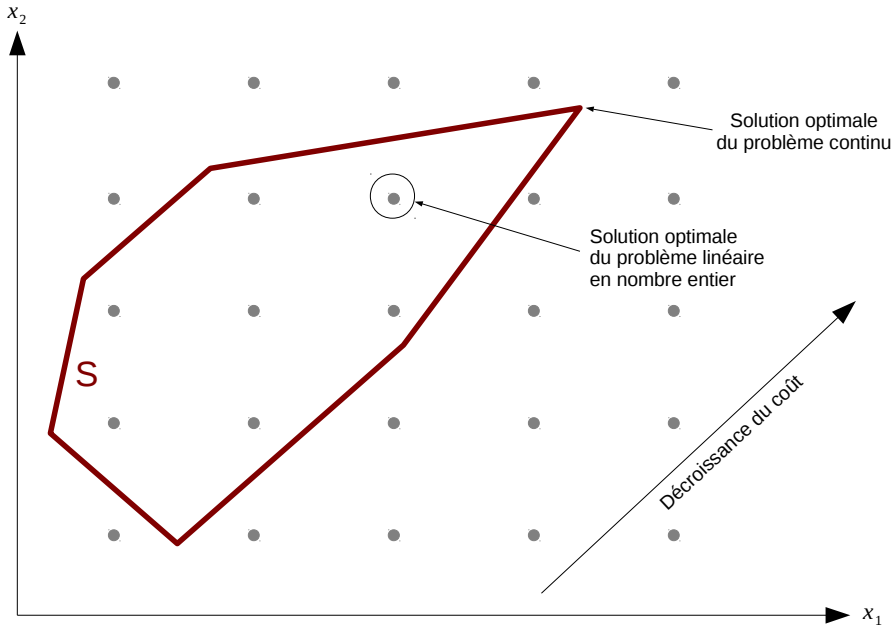


FIGURE 3.3 – Illustration de la difficulté liée aux problèmes d'optimisation en nombre entiers

L'exploration exhaustive de l'ensemble des solutions étant clairement impossible au delà d'une toute petite taille de problème (en raison de l'explosion combinatoire), des méthodes avancées ont vu le jour pour explorer intelligemment l'espace de solutions.

La méthode appelée B&B (pour Branch and Bound, en français séparation et évaluation) est l'une des plus utilisées pour la recherche de solutions contenant des variables entières. Ce type de méthode consiste à représenter l'espace des solutions par une arborescence qu'il faut parcourir le plus intelligemment possible pour arriver à la solution. Plus précisément, l'algorithme B&B cherche à éliminer progressivement l'exploration des solutions dont on sait d'ores et déjà qu'elle ne permettront pas d'obtenir une meilleure solution que la solution courante. Nous illustrons son fonctionnement sur la Figure 3.4.

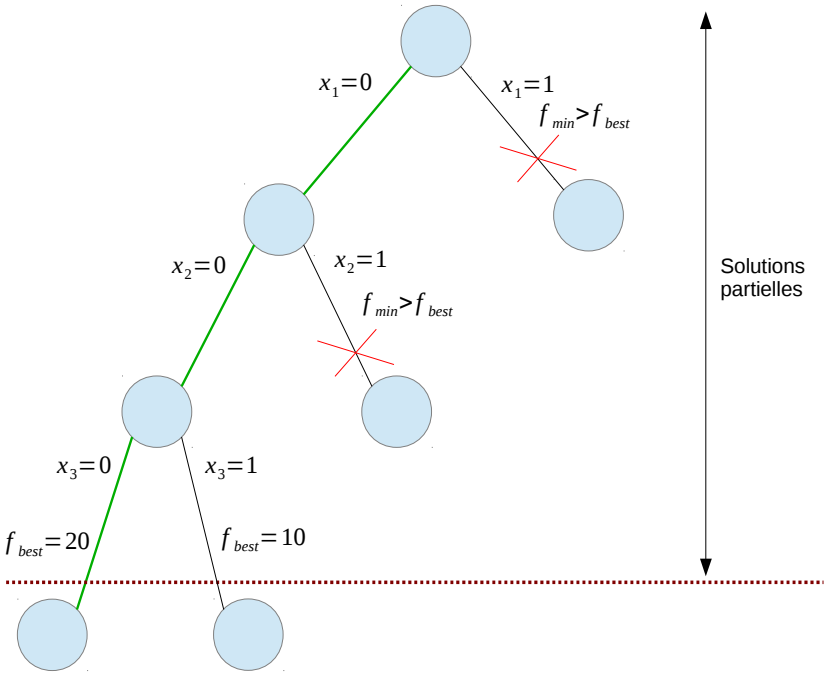


FIGURE 3.4 – Illustration du branch and bound

Chaque noeud correspond à une solution partielle pour laquelle une partie des variables a des valeurs entières fixées, les autres valeurs étant alors encore à déterminer. Dans cet exemple, l'algorithme démarre à la racine de l'arborescence, pour laquelle aucune variable n'a encore été fixée. Le coût du vecteur nul ($x_i = 0, \forall i$) est connu et vaut $f_{best} = 20$. Pour chaque noeud traversé dans l'arborescence, une borne inférieure sur le coût f_{min} est calculée à partir de la solution entière partielle (valeurs fixées à partir des noeuds précédents), tout en relaxant les contraintes de nombres entiers pour les valeurs qui n'ont pas encore été fixées. Les branches de l'arborescence pour lesquelles la borne inférieure f_{min} est supérieure à la borne supérieure f_{best} trouvée précédemment sont sup-

primées et donc jamais parcourues. Si lorsqu'une solution complète est générée, son coût est inférieur à la borne supérieure f_{best} courante, alors la borne supérieure est mise à jour. Dans cet exemple, les branches où $x_1 = 1$ et où $x_1 = 0, x_2 = 1$ ne sont pas parcourues, et l'algorithme trouve immédiatement la meilleure solution $x_1 = 0, x_2 = 0, x_3 = 1$.

L'algorithme B&B est aujourd'hui intégré dans tous les grands solveurs de problèmes linéaires en nombres entiers. Il permet d'améliorer considérablement les temps de calculs nécessaires vis à vis d'une exploration exhaustive. Malgré tout, l'algorithme nécessite des appels répétés à la fonction calculant la borne inférieure f_{min} en chaque nœud, et ces calculs peuvent s'avérer rapidement coûteux en temps. Les tailles de problèmes soumis à la méthode B&B se doivent donc de rester limitées pour éviter l'explosion des temps de calcul.

3.3 Optimisation non-linéaire

Les méthodes classiquement utilisées pour la résolution de problèmes linéaires continus, tels que l'algorithme du simplexe, ne peuvent être utilisées dès lors que l'un des éléments du problème (fonction coût ou contrainte) est non linéaire. Nous présentons ici des méthodes itératives à direction de descente, basées sur l'utilisation du gradient, qui permettent de résoudre ces problèmes. Ces méthodes génèrent une suite de points (x_k) convergeant vers un optimum local de la fonction coût f. Nous commençons par présenter les méthodes du gradient et du gradient conjugué, qui permettent de résoudre des problèmes d'optimisation non linéaires, continus et sans contraintes. Nous abordons ensuite la méthode du gradient projeté qui permet de prendre en compte des contraintes.

3.3.1 Méthode du gradient

La méthode du gradient permet de résoudre des problèmes non linéaires sans contraintes. L'idée de la méthode consiste pour chaque nouveau point x_k, à emprunter une direction de descente d_k dans le sens contraire du gradient, pour faire décroître la fonction coût :

$$d_k = -\nabla f(x_k) \tag{3.4}$$

Le prochain point est alors déterminé suivant la direction de descente :

$$x_{k+1} = x_k + \alpha_k d_k \tag{3.5}$$

où le pas $\alpha_k \in \mathbb{R}$ est solution du problème suivant :

$$f(x_k + \alpha_k d_k) = \min_{\alpha \in \mathbb{R}} f(x_k + \alpha d_k) \tag{3.6}$$

α_k est donc le pas de descente permettant de minimiser la fonction coût dans le sens de

la direction de descente. L'algorithme itère ainsi jusqu'à ce qu'un point stationnaire soit trouvé. Ce point stationnaire correspond alors un point où le gradient est nul ($\nabla f(x) = 0$). Nous illustrons un exemple de chemin suivi par l'algorithme sur la figure 3.5. Le gradient, et donc la direction de descente suivie, sont toujours orthogonaux à la tangente de la ligne iso-coût au point courant.

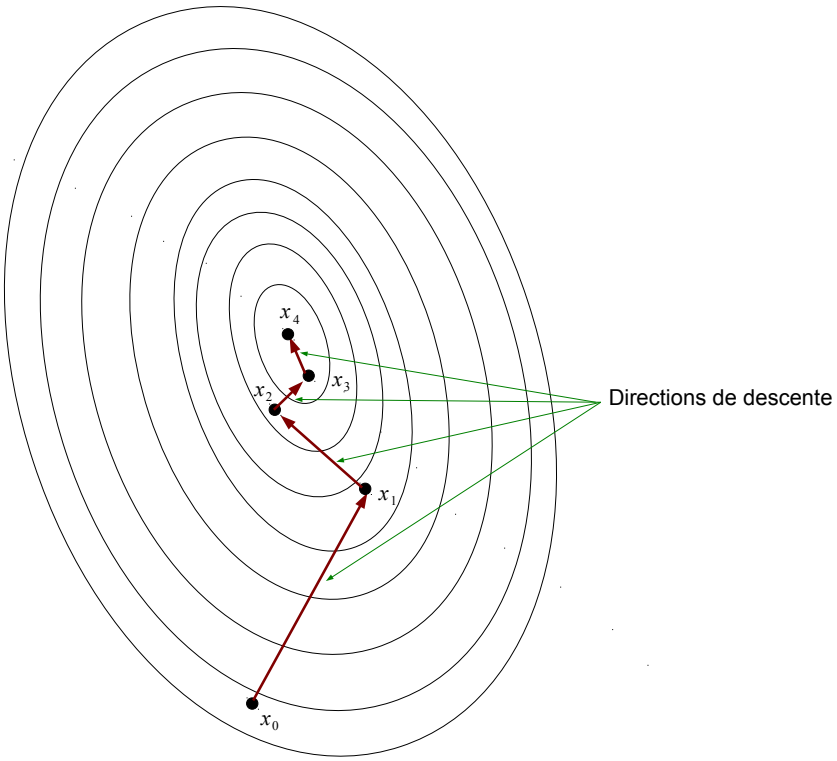


FIGURE 3.5 – Illustration de la méthode du gradient

La simplicité de mise en œuvre de l'algorithme est contrebalancée par sa relative lenteur de convergence : l'algorithme peut nécessiter un grand nombre d'itérations avant de converger vers un minimum local. De plus, la recherche du paramètre α_k est généralement mise en œuvre sous forme d'une recherche linéaire et peut aussi s'avérer coûteuse en temps.

3.3.2 Méthode du gradient conjugué

La méthode du gradient conjugué permet souvent d'accélérer fortement la convergence par rapport à la méthode classique du gradient. Elle modifie les directions de descente utilisées, en considérant plus d'informations sur la fonction à optimiser. Les directions suivies sont dites mutuellement conjuguées. Les points parcourus sont toujours

de la forme :

$$x_{k+1} = x_k + \alpha_k d_k \tag{3.7}$$

mais la direction de descente est ici une combinaison linéaire du gradient au point courant et des directions de descente précédentes :

$$d_{k+1} = -\nabla f(x_{k+1}) + \beta_k d_k \tag{3.8}$$

où le paramètre β_k peut être calculé suivant différentes formules. La première formulation fut proposée par Fletcher et Reeves dans [44], et porte le nom de ces auteurs :

$$\beta_k = \frac{\nabla f(x_{k+1}) \nabla f(x_{k+1})}{\nabla f(x_k) \nabla f(x_k)} \tag{3.9}$$

Une autre formulation, qui est souvent privilégiée pour les fonctions non quadratiques, fut proposée par Polak et Ribiere [94] :

$$\beta_k = \frac{\nabla f(x_{k+1}) \left(\nabla f(x_{k+1}) - \nabla f(x_k)\right)}{\nabla f(x_k) \nabla f(x_k)} \tag{3.10}$$

3.3.3 Méthode du gradient projeté

Cette méthode adapte la méthode du gradient pour qu'elle prenne en compte des contraintes. L'idée de la méthode consiste à projeter la direction de descente sur l'espace des contraintes, à chaque itération de l'algorithme. Elle chemine donc le long de la frontière de l'espace des contraintes.

Si l'on définit P_S comme l'opérateur de projection sur l'espace des solutions admissibles S :

$$P_S(x) = \underset{y \in S}{\operatorname{argmin}} \, ||x - y||^2 \tag{3.11}$$

alors la suite de points du gradient projeté est définie par le relation suivante :

$$x_{k+1} = x_k - \alpha_k P_S \left[\nabla f(x_k)\right] \tag{3.12}$$

Le pas α_k est alors déterminé pour que le nouveau point reste dans S, tout en diminuant la valeur de la fonction coût. Cette méthode est efficace à condition que l'opérateur de projection reste suffisamment simple. L'un des premiers algorithmes construit sur ce principe fut proposé par Rosen dans [99].

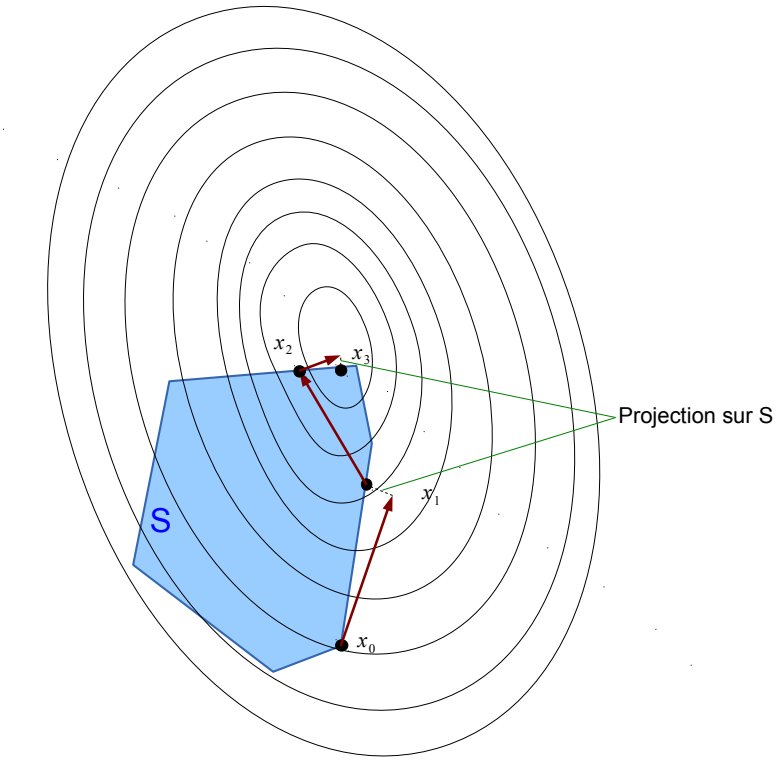


FIGURE 3.6 – Illustration de la méthode du gradient projeté

3.4 Méthodes heuristiques et métaheuristiques

Les méthodes exactes de résolution sont lourdes en calcul et sensibles au passage à l'échelle. Les temps de calcul sont la plupart du temps exponentiels avec la taille du problème considéré. Ces méthodes sont donc surtout utilisées pour fournir des preuves d'optimalité, sur des problèmes de taille limitée.

Les difficultés de mise à l'échelle justifient l'existence d'heuristiques et de méthodes d'approximations, conçues pour trouver de bonnes solutions avec une moindre complexité de calcul. Contrairement à une heuristique, un algorithme d'approximation est une méthode de résolution approchée pour laquelle il existe une garantie sur la qualité de la solution obtenue.

Les méta-heuristiques sont des méthodes d'optimisation itératives, qui tentent de progresser vers un optimum local ou global. Parmi les méta-heuristiques, on peut citer la recherche locale (cf. 3.4.2), les algorithmes de colonie de fourmis (cf section 3.4.3), les algorithmes génétiques [77], ou encore les algorithmes de type recuit simulé [40].

Nous présentons dans les sections suivantes certaines de ces techniques, qui seront utilisées dans les prochains chapitres de ce manuscrit.

3.4.1 Algorithme glouton

Les algorithmes gloutons ("greedy algorithms" en anglais) sont des algorithmes très simples qui construisent une solution de manière incrémentale. Pour chaque point courant, une fonction de sélection, nommée également "critère glouton" est utilisée pour choisir le prochain point. Le critère de sélection est le critère primordial de la méthode, car il détermine quel est le prochain point localement optimal. Si l'approche fournit l'optimum global, on parle d'algorithme glouton exact (par exemple pour la recherche d'un arbre couvrant minimum dans un graphe [66]). Mais dans la plupart des cas, il ne fournit qu'une approximation, potentiellement très éloignée de l'optimum global : on parle alors d'heuristique gloutonne. La grande force de ces méthodes repose sur leur grande rapidité d'exécution.

Lorsque le temps exécution doit rester limité, l'utilisation de ce type de méthode, en association avec un bon choix de critère glouton, peut fournir de très bons résultats. Dans ce mémoire, nous mettons ainsi en oeuvre une méthode gloutonne pour l'optimisation des poids OSPF (cf chapitre 4)

3.4.2 Recherche locale

La recherche locale est une méta-heuristique itérative permettant d'optimiser des problèmes combinatoires difficiles. Il s'agit d'un processus itératif dans lequel on cherche à améliorer la solution courante x_k en explorant son voisinage $N(x_k)$, k étant l'itération courante. Le voisinage N est obtenu en considérant une transformation élémentaire de la solution courante, et sa définition exacte dépend du problème précis à résoudre.

Il existe plusieurs possibilités pour choisir la prochaine solution dans le voisinage. La première approche consiste à choisir la première solution rencontrée lors de l'exploration qui améliore le coût courant. Cette façon de faire est connue dans la littérature sous le nom de "first-descent". Une deuxième méthode, plus coûteuse en temps de calcul, mais qui fournit généralement de meilleurs résultats, consiste à rechercher le meilleur voisin possible dans l'espace N. Cette méthode, que l'on nomme "steepest-descent", est décrite dans l'Algorithme 1.

La recherche locale procède de cette manière jusqu'à ce qu'aucun voisin ne puisse fournir de meilleure solution que la solution courante. Il s'agit d'une méthode permettant de trouver un minimum local : l'algorithme peut potentiellement rester coincé sur un minimum local éloigné de l'optimum global. Certaines modifications apportées à l'algorithme permettent de limiter ce désagrément : la première d'entre elles consiste à considérer plusieurs optimisations successives en démarrant avec des points aléatoires distincts [72]. Une deuxième approche envisageable consiste à autoriser temporairement (pour un certain nombre maximum d'itérations) une augmentation du coût lorsqu'aucune solution dans le voisinage ne permet de l'améliorer.

Algorithme 1 Recherche locale de type "steepest-descent"

choisir un point initial x
repeat
  $z \leftarrow x$
  for $y \in N(x)$ **do**
    if $f(y) < f(z)$ **then**
      $z \leftarrow y$
    end if
  end for
until $z = x$.
return x

3.4.3 Optimisation par colonie de fourmis

Les algorithmes de colonies de fourmis sont des métaheuristiques basées sur l'observation de vraies colonies de fourmis qui furent menées par des biologistes [36], et sur la formalisation faite par Marco Dorigo dans [38] et [39]. Ces métaheuristiques reproduisent le comportement collectif des fourmis, qui utilisent une forme d'intelligence collective basée sur un dépôt de phéromones. Chaque individu explore l'espace de recherche d'une manière aléatoire qui dépend directement du taux de phéromones déposés par ses congénères. Ces techniques sont reconnues comme pouvant être efficaces sur des problèmes nécessitant de trouver de bons chemins dans un graphe. Nous aborderons cette méthode plus en détails dans le chapitre 5, où nous avons pu utiliser un algorithme qui s'en inspire directement.

4

Optimisation dynamique des réseaux OSPF

4.1 Introduction

Avec la popularité croissante des applications gourmandes en bande passante (streaming multimédia, vidéo à la demande, partage de fichier pair à pair [102], etc...), les trafics réseau deviennent de plus en plus volatiles. Une conséquence de cette haute variabilité du trafic est qu'il n'est plus crédible de concevoir un réseau uniquement sur la base d'une matrice de trafic en "heure de pointe". Une telle approche peut mener à de mauvaises performances du réseau si à un moment donné la matrice de trafic s'éloigne suffisamment de celle utilisée pour l'ingénierie du trafic. Une alternative bien connue pour gérer les variations de trafic consiste à superviser dynamiquement le trafic, pour adapter l'utilisation des ressources lorsque des changements sont observés. Un des mécanismes fondamentaux pour contrôler les performances réseau est l'optimisation du routage, qui permet d'utiliser plus efficacement les ressources en adaptant les routes au trafic courant transporté. Cependant, plusieurs difficultés émergent lorsque l'on cherche à concevoir et implémenter un routage adaptatif dans les réseaux IP.

La première difficulté est liée aux protocoles de routage intra-domaine (IGP), les plus utilisés étant OSPF et IS-IS [78, 80]. Chaque flux de trafic y est routé le long du plus court chemin, tout en procédant à un partage de charge équitable des flux lorsque plusieurs liens sortants d'un nœud se trouvent sur des plus courts chemins jusqu'à la destination. Bien qu'habituellement mis à 1, le poids des liens, et donc les plus courts chemins, peuvent être changés par l'opérateur réseau. Si l'on considère un ensemble de demandes de trafic entre des paires origine/destination (OD), le problème d'optimisation

des poids des liens consiste à trouver l'ensemble de poids des liens optimisant une mesure de performance donnée, telle que le taux maximum d'utilisation des liens (voir [49, 46, 114, 50]).

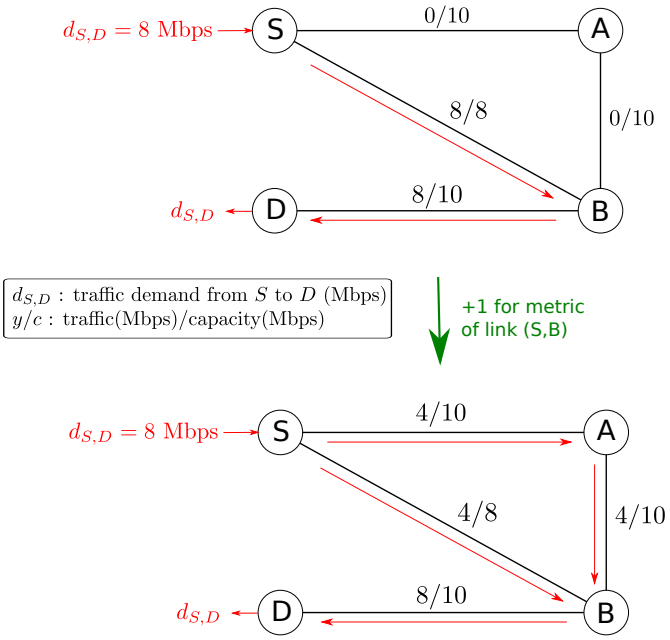


FIGURE 4.1 – Exemple d'optimisation de métriques OSPF.

Un exemple est présenté en Figure 4.1, où un unique flux de demande 8 Mbps doit être routé du nœud S vers le nœud D. Toutes les capacités des liens sont fixées à 10 Mbps, à l'exception du lien S-B dont la capacité vaut 8 Mbps.

Si des poids unitaires sont utilisés pour chaque lien, le flux est routé sur le chemin S-B-D, ce qui conduit à un taux maximum d'utilisation sur le réseau égal à 100%. Mais, si le poids du lien S-B est incrémenté de 1, la moitié du trafic est dévié sur le chemin S-A-B-D, réduisant ainsi le taux maximum d'utilisation à 80% sur le lien B-D (tandis que les autres liens restent congestionnés à moins de 50%).

On peut noter que le problème d'optimisation des poids des liens est connu comme étant NP-difficile [48].

La deuxième difficulté réside dans la surveillance des demandes des trafics origine-destination. Les méthodes traditionnelles d'optimisation des poids de liens ont été conçues à des fins de planification : elles supposent qu'une demande en trafic est *prédite* pour chaque paire origine - destination dans le réseau, de manière exacte ou bien avec une forme d'incertitude [81, 12, 13, 29]. En pratique, les demandes en trafic ne peuvent pas être directement mesurées dans les réseaux de grande taille en raison du surcoût important induit par l'utilisation des solutions actuelles de surveillance/mesure réseau telles que Netflow [32, 31]. Une approche alternative consiste à utiliser les "links counts"

(quantité de trafic transmise sur un lien pendant une période de 5 minutes) fournis par le protocole Simple Network Management Protocol (SNMP) afin de déterminer une approximation des demandes transitant réellement sur le réseau. Mais, comme il y a souvent plus de flux réseau que de mesures de charge sur les liens, cela revient à considérer un problème inverse mal posé qui ne peut être résolu sans information supplémentaire (voir par exemple [121, 106, 84, 47, 107, 89] et les références qui s'y trouvent). Il est également important de noter que ces mesures fournissent une forme d'information sur des trafics passés, alors que les décisions de routage doivent être prises pour les demandes futures.

Enfin, il existe certaines difficultés liées aux contraintes opérationnelles. La première d'entre elles est que les changements de routage doivent être limités en nombre, pour éviter de changer continuellement les routes du réseau. Ces modifications doivent aussi être incrémentales : la stratégie de routage courante doit être améliorée de façon incrémentale en modifiant le poids d'un seul lien à la fois. De plus, les décisions de reconfiguration doivent être prises en temps-réel. Si l'on considère que les mesures de charge sur les liens sont disponibles toutes les 5 minutes, et que les protocoles de routage intra-domaines requièrent quelques dizaines de secondes pour converger vers les nouvelles routes, cela signifie que le processus de décision ne devrait durer que quelques secondes. Ces contraintes opérationnelles ont un impact important sur la complexité de l'algorithme d'optimisation que l'on peut utiliser.

Dans ce chapitre, nous nous intéressons à une méthode pour dynamiquement reconfigurer les poids des liens pour les adapter au trafic courant. Un problème similaire a été considéré dans [119], mais, contrairement au travail présenté ici, les auteurs supposent que la moyenne et la variance des trafics agrégés entre chaque routeur source et chaque routeur destination est disponible périodiquement.

Une autre référence étroitement liée est [24], où les auteurs considèrent le même problème que nous, mais proposent une approche ne pouvant faire face à des contraintes temps-réel.

Nous proposons un algorithme de reconfiguration dynamique des routes intra-domaine pour optimiser les charges des liens dans les cœurs de réseaux IP. Cet algorithme utilise le protocole SNMP pour collecter périodiquement des informations de charges des liens, à partir desquelles un ensemble de matrices de trafic compatibles est déterminé. Une heuristique d'optimisation robuste est ensuite utilisée pour minimiser le taux de congestion du réseau, i.e., le taux d'utilisation du lien le plus congestionné. Les résultats de simulation, comme les résultats obtenus à partir de vraies données de trafic réseau montrent que la méthode proposée, malgré sa simplicité apparente, autorise une importante amélioration des performances du réseau, tout en conservant des temps d'exécution compatibles avec un fonctionnement en ligne.

On note que l'algorithme dynamique proposé peut aider à limiter les effets d'oscillation de routes (route flapping) dans les réseaux OSPF. Si un routeur OSPF ne reçoit pas

4 paquets *hello* consécutifs depuis un voisin, il considère que ce voisin n'est plus accessible (lien mort) et purge de sa table de routage toutes les routes qui étaient accessibles via ce voisin. Un lien ne répondant plus à cause d'une congestion trop importante (tous les paquets hello sont perdus) peut engendrer un transfert de trafic massif d'une route vers une autre. Le lien d'origine se stabilise alors car le trafic (et donc la congestion) s'est déplacé vers un autre lien, et redevient rapidement disponible. Alors qu'il récupère à nouveau son trafic d'origine, il redevient à nouveau congestionné, ce qui recommence le cycle. Ce phénomène de transfert répété des trafics peut sembler sans solution évidente. Le processus dynamique que nous proposons pourrait aider à limiter les effets de ce phénomène en répartissant la charge réseau entre les routes avant qu'une congestion réseau n'apparaisse.

Le chapitre est organisé comme suit. La section 4.2 est consacrée à la formulation mathématique du problème. L'algorithme proposé et ses détails sont décrits dans la section 4.3. Les résultats sur des trafics simulés et sur des trafics mesurés sur un réseau opérationnel sont exposés dans la section 4.4. Enfin, des conclusions sont présentées dans la section 4.5.

4.2 Définition du problème

Le réseau est représenté par le graphe $G = (V, E)$. L'ensemble V est composé des N nœuds du réseau, tandis que l'ensemble E se compose des M liens du réseau. On note c_l la capacité du lien l, et $K = N(N-1)$ le nombre de paires origine - destination (OD).

Le réseau est observé aux instants discrets $\tau = 1, 2, \ldots$ Soit $\hat{\mathbf{y}}^\tau = (\hat{y}_1^\tau, \ldots, \hat{y}_M^\tau)$ le vecteur des mesures de trafic sur les liens, où $\hat{y}_l^\tau$ indique le trafic moyen ayant transité sur le lien l entre les temps $\tau - 1$ et τ. Ces mesures fournissent une observation indirecte sur les trafics moyens de bout en bout durant cet intervalle de temps.

Même si les demandes en trafic peuvent naturellement être représentées sous forme matricielle, nous choisissons ici d'utiliser une notation vectorielle. Nous fixons arbitrairement un ordre sur les demandes et notons d_k^τ le trafic moyen transmis par la paire OD k durant l'intervalle de temps $\mathcal{I}_\tau = [\tau - 1, \tau]$. Soient $s(k)$ et $t(k)$ les nœuds source et destination de la demande k. Le vecteur des demandes est représenté par $\mathbf{d}^\tau$. Nous rappelons que les demandes en trafic doivent ici être interprétées comme des trafics offerts, autrement dit le trafic qui serait transporté si les capacités des liens étaient infinies. En effet, lorsque l'on évalue les bénéfices attendus en déviant un flux d'un lien, il est plus logique de raisonner en termes de trafic offert. Il est bon de noter qu'en conséquence le taux d'utilisation des liens peut devenir (du moins théoriquement) supérieur à 100%.

On peut contrôler le réseau en changeant les poids des liens. Soit $\boldsymbol{\omega}^\tau = (\omega_1^\tau, \ldots, \omega_M^\tau)$ le vecteur décrivant la configuration de tous les poids durant l'intervalle de temps $\mathcal{I}_\tau$, où le poids ω_l^τ du lien l est une valeur entière comprise dans l'intervalle $\Omega = \left[1, 2^{16} - 1\right]$. Les plus courts chemins résultant de la configuration des poids $\boldsymbol{\omega}^\tau$ sont modélisés par

un matrice de routage $\mathbf{F}(\boldsymbol{\omega}^\tau)$ de taille $M \times K$, dont les lignes représentent les liens du réseau et dont les colonnes représentent les paires OD. L'élément $f_{l,k}(\boldsymbol{\omega}^\tau)$ correspond à la fraction du trafic k routée sur le lien l. Ces valeurs sont directement obtenues à partir du vecteur des poids de liens en utilisant n'importe quel algorithme de plus court chemin, tel que l'algorithme de Dijkstra. Nous rappelons que ces fractions de trafic ne sont pas contraintes à 0 ou à 1 en raison de la répartition du trafic sur les chemins d'égale longueur.

Les demandes OD et les trafics observés sur les liens sont alors liés par l'équation linéaire suivante :

$$\hat{\mathbf{y}}^\tau = \mathbf{F}(\boldsymbol{\omega}^\tau)\, \mathbf{d}^\tau. \tag{4.1}$$

En modifiant la configuration des poids, nous pouvons contrôler la matrice de routage et donc les charges de liens qui en résultent. La principale difficulté est d'arriver à choisir une configuration de poids $\boldsymbol{\omega}^\tau$ à l'instant τ sans aucun moyen de prédire les demandes en trafic futures entre les instants τ et $\tau+1$. Comme nous ne disposons d'aucune information sur les trafics futurs, notre approche consiste à réagir à la congestion réseau observée en optimisant la configuration des poids pour les trafics actuels $\mathbf{d}^\tau$. En un sens, nous procédons comme si $\mathbf{d}^{\tau+1} = \mathbf{d}^\tau$. L'idée est que, même si cette hypothèse n'est pas complètement respectée, il existe une forme de stabilité dans le trafic permettant de prendre de bonnes décisions en utilisant la connaissance la plus récente du trafic.

Le problème que nous considérons consiste à trouver la configuration des poids $\boldsymbol{\omega}^\tau$ minimisant le taux de congestion du réseau (défini comme le taux maximum d'utilisation de tous les liens) pendant la période $\mathcal{I}_\tau$. Plus formellement, le problème s'exprime ainsi :

$$\begin{aligned} &\text{minimiser } \rho(\boldsymbol{\omega}) = \max_{l \in E} \frac{y_l}{c_l} && \text{(METRIC)} \\ &\text{avec} \\ &\qquad \mathbf{y} = \mathbf{F}(\boldsymbol{\omega})\, \mathbf{d}^\tau, \\ &\qquad \boldsymbol{\omega} \in \Omega^M. \end{aligned}$$

On remarque que la solution du problème d'optimisation ci-dessus dépend de la matrice de trafic à l'instant τ. Ces trafics doivent être déduits des observations disponibles. Nous décrivons l'approche proposée pour résoudre ce problème dans la section suivante.

4.3 Algorithme en ligne

L'algorithme en ligne que nous proposons pour la reconfiguration dynamique des routes IP est décrit en Figure 4.2. Cet algorithme est exécuté périodiquement. Il commence par utiliser le protocole SNMP pour collecter des informations sur le trafic moyen sur chaque lien durant la dernière fenêtre temporelle. Ces observations sont ensuite utilisées pour estimer un ensemble d'incertitude sur la demande, sur lequel l'algorithme d'optimisation doit s'appuyer. Une heuristique d'optimisation gloutonne est utilisée pour déterminer si certaines modifications de poids de liens doivent être appliquées pour réduire la congestion réseau. Si aucun changement de poids intéressant n'est trouvé, l'algorithme s'arrête jusqu'à son réveil à la prochaine période de temps. Sinon, les changements de poids sont appliqués sur le réseau, avant de s'arrêter, dans l'attente des prochaines mesures SNMP. Nous décrivons dans ce qui suit les principales étapes de l'algorithme.

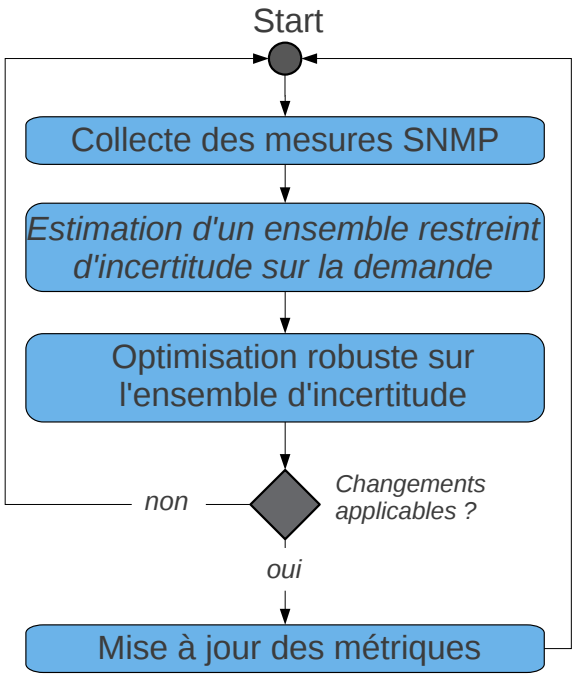


FIGURE 4.2 – Algorithme en ligne d'optimisation des poids OSPF.

Remarque 4.3.1. *SNMP est un protocole Internet pour gérer les périphériques sur les réseaux IP, tels que des routeurs. La variable MIB nommée ifOutOctets, qui est utilisée pour calculer les link counts (nombre d'octets transmis sur l'interface réseau), est directement appliquée par l'agent SNMP s'exécutant au niveau du routeur pour l'ensemble de ses interfaces. Nous considérons donc cette valeur comme fiable et précise. Le temps nécessaire pour collecter l'ensemble des link counts via le protocole SNMP est difficile*

à évaluer précisément, car il dépend de multiples facteurs, mais il doit être de l'ordre de quelques secondes (dépendant de la taille du réseau, de l'architecture de l'outil de management réseau, etc. . .). Le surcoût en trafic doit pouvoir être négligé, et, comme SNMP est un processus de basse priorité pour l'ordonnancement des tâches du routeur, l'agent SNMP ne devrait pas en affecter les performances.

4.3.1 Estimation de la matrice de trafic

Pour résoudre le problème (METRIC), nous devons connaître les demandes en trafic à l'instant τ. Dans la pratique, ces demandes en trafic sont inconnues et doivent être déduites des mesures SNMP disponibles. Nous supposons qu'à l'instant τ nous obtenons les informations suivantes concernant les demandes en trafic sur la période de temps $\mathcal{I}_\tau$:

— le trafic moyen $\hat{y}_l^\tau$ sur chaque lien $l \in E$,

— le trafic moyen entrant $b_n^{i,\tau}$ et le trafic moyen sortant $b_n^{e,\tau}$ pour chaque routeur de bordure n.

Le vecteur $\mathbf{d}^\tau$ des demandes en trafic sur la période $\mathcal{I}_\tau$ est relié au vecteur $\hat{\mathbf{y}}^\tau$ des trafics observés sur les liens par l'intermédiaire de l'équation (4.1). Même si la matrice de routage $\mathbf{F}(\boldsymbol{\omega}^\tau)$ est connue à l'instant τ, cela n'est généralement pas suffisant pour déterminer les demandes en trafic. En effet, il s'agit d'un problème inverse mal posé car dans la grande majorité des réseaux, le nombre K de paires OD est plus grand que le nombre de liens M. Malgré tout, nous pouvons définir le polytope $\mathcal{D}^\tau$ des matrices de trafic respectant les observations comme l'ensemble des vecteurs $\mathbf{d} \in \mathbb{R}_+^K$ tels que

$$\hat{\mathbf{y}}^\tau = \mathbf{F}(\boldsymbol{\omega}^\tau)\,\mathbf{d}, \tag{4.2}$$

$$\sum_{k:s(k)=n} d_k = b_n^{i,\tau}, \quad \forall n \in V, \tag{4.3}$$

$$\sum_{k:t(k)=n} d_k = b_n^{e,\tau}, \quad \forall n \in V. \tag{4.4}$$

Comme nous ne pouvons pas déterminer exactement les demandes en trafic à l'instant τ, nous devons utiliser des informations additionnelles pour les estimer. Les méthodes existantes pour l'estimation de matrice de trafic diffèrent par la nature des informations additionnelles utilisées.

Impulsé par [117], une première génération de méthodes a tenté d'utiliser les covariances des charges de liens comme information supplémentaire (voir par exemple, [74, 26, 62]). Ces méthodes sont basées sur des hypothèses statistiques sur les trafics dont la validité peut être remise en cause. Une seconde génération de méthodes utilise des informations spatiales ou temporelles a priori [122, 106, 84, 47]. La dernière génération de méthodes utilise des informations spatiales et temporelles a posteriori. Elles

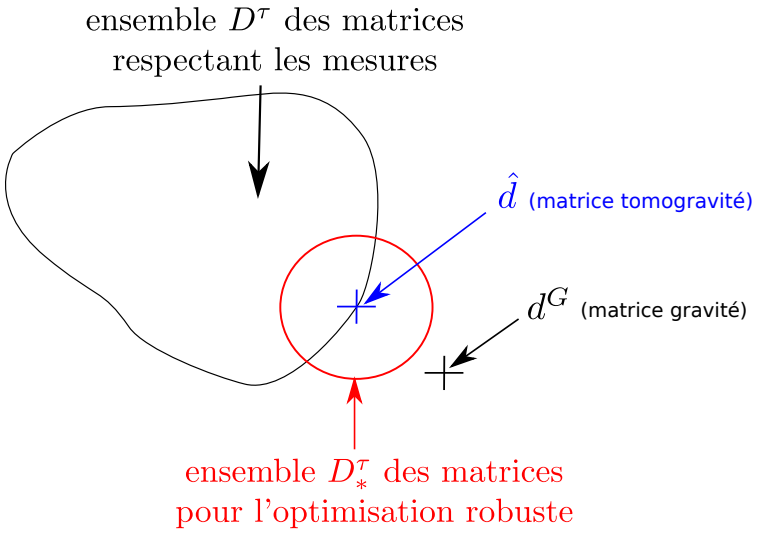


FIGURE 4.3 – Matrices et ensemble d'incertitude utilisés pour l'optimisation dynamique des poids OSPF.

requièrent une phase de calibration où les matrices de trafic sont régulièrement mesurées en utilisant Netflow sur une certaine période de temps, puis elles utilisent des techniques de filtrage pour prédire les demandes en trafic futures [107, 89]. Même si la précision de ces méthodes est supérieure à celle des premières générations de méthodes, elles impliquent un surcoût important en mesure durant la phase de calibration. Nous avons choisi d'utiliser la méthode nommé "tomogravité", introduite dans [122], car il s'agit de la plus simple et de la plus rapide des méthodes de deuxième génération, qu'elle ne repose sur aucune hypothèse statistique sur le trafic (à l'inverse des méthodes de première génération), et qu'elle n'introduit pas de surcoût en trafic sur le réseau (à l'inverse des méthodes de troisième génération). L'idée principale de la méthode revient à considérer le vecteur de trafic $\mathbf{d}^G$ obtenu en utilisant la formulation suivante :

$$d_k^G = \frac{b_{t(k)}^{e,\tau}}{\sum_{n \neq s(k)} b_n^{e,\tau}} b_{s(k)}^{i,\tau}. \tag{4.5}$$

En d'autres mots, le vecteur de trafic $\mathbf{d}^G$ est obtenu en supposant que le trafic total entrant dans un nœud est divisé entre toutes les demandes provenant de ce nœud, en proportion du trafic total sortant des nœuds destination.

En général, les demandes en trafic d_k^G obtenues ne respectent pas les observations, i.e., $\mathbf{d}^G \notin \mathcal{D}^\tau$. Pour obtenir des demandes respectant les observations, la tomogravité calcule la projection du vecteur de trafic $\mathbf{d}^G$ sur le polytope $\mathcal{D}^\tau$ (voir Figure 4.3). Le vecteur de trafics estimé $\hat{\mathbf{d}}$ est alors le vecteur appartenant à $\mathcal{D}^\tau$ minimisant la distance

à $\mathbf{d}^G$. En choisissant la norme infinie, le vecteur de trafic $\hat{\mathbf{d}}$ est alors la solution du problème d'optimisation linéaire suivant :

$$\begin{array}{ll} \text{minimiser} & \alpha \\ \text{avec} & \\ & \alpha \geq |\hat{d}_k - d_k^G| \qquad\qquad k = 1, \ldots, K \\ & \hat{y}_l^\tau = \sum_{k=1}^{K} f_{l,k}(\boldsymbol{\omega}^\tau)\hat{d}_k, \quad \forall l \in E \\ & \sum_{k:s(k)=n} \hat{d}_k = b_n^{i,\tau}, \qquad \forall n \in V \\ & \sum_{k:t(k)=n} \hat{d}_k = b_n^{e,\tau}, \qquad \forall n \in V \end{array}$$

On note que la valeur minimale de α doit être interprétée comme la distance entre le vecteur de tomogravité $\mathbf{d}^G$ et le polytope des matrices respectant les mesures SNMP.

Notre algorithme en ligne utilise la bibliothèque logicielle CPLEX (C++) [2] pour calculer le vecteur de trafic $\hat{\mathbf{d}}$ dès que les mesures SNMP sont disponibles. Les demandes en trafic estimées $\hat{d}_k$ peuvent être obtenues très efficacement, en quelques fractions de secondes pour de petits réseaux, et quelques secondes à peine pour les réseaux de plus grande taille. Ces demandes estimées sont alors utilisées pour optimiser les poids des liens pour la période de temps suivante $\mathcal{I}_{\tau+1}$.

Si une modification de topologie (un lien apparaissant/disparaissant par exemple) intervient entre les instants $\tau-1$ et τ, le problème de minimisation ci-dessus peut devenir insoluble. Dans un tel cas, pour éviter de prendre de mauvaises décisions en raisons de données non cohérentes, aucune modification de poids ne sera appliquée, l'algorithme se contenant alors d'attendre la prochaine salve de mesures SNMP.

4.3.2 Optimisation robuste des poids des liens

Pour prendre en compte l'erreur d'estimation sur le vecteur $\hat{d}_k$ et l'incertitude sur les variations des trafics pendant la période $\mathcal{I}_\tau$, nous faisons une hypothèse de variation bornée des trafics. Nous supposons que la demande d'un flux pendant la période $\mathcal{I}_\tau$ vérifie l'hypothèse suivante :

$$(1-\gamma)\,\hat{d}_k \leq d_k^\tau \leq (1+\gamma)\,\hat{d}_k, \quad \forall k = 1, \ldots, K, \tag{4.6}$$

où le paramètre γ est fixé. Nous supposons que la totalité des trafics entrant et sortant de chaque nœud se matérialisent par des demandes transitant sur le réseau :

$$\sum_{k:s(k)=n} d_k^\tau = b_n^{i,\tau}, \quad \forall n \in V, \tag{4.7}$$

$$\sum_{k:t(k)=n} d_k^\tau = b_n^{e,\tau}, \quad \forall n \in V. \tag{4.8}$$

On note $\mathcal{D}_*^\tau$ l'ensemble des vecteurs de trafic $\mathbf{d}$ respectant (4.6)-(4.8). L'ensemble $\mathcal{D}_*^\tau$ décrit l'incertitude sur la demande pour l'instant τ. Comme nous l'expliquons dans la section 4.3.2.2, le principal avantage de l'utilisation du polytope $\mathcal{D}_*^\tau$ au lieu de $\mathcal{D}^\tau$ est qu'il simplifie grandement le calcul des charges des liens au pire cas. En définissant $y_l(\boldsymbol{\omega}, \mathbf{d}) = \sum_k f_{l,k}(\boldsymbol{\omega})\, d_k$ comme le trafic sur le lien l avec la configuration de poids $\boldsymbol{\omega}$ quand le vecteur de trafic est $\mathbf{d}$, et en notant que

$$\begin{aligned} \rho(\boldsymbol{\omega}) &= \max_{\mathbf{d}\in\mathcal{D}_*^\tau} \max_{l\in E} \frac{1}{c_l} y_l(\boldsymbol{\omega}, \mathbf{d}), \\ &= \max_{l\in E} \max_{\mathbf{d}\in\mathcal{D}_*^\tau} \frac{1}{c_l} y_l(\boldsymbol{\omega}, \mathbf{d}), \end{aligned} \tag{4.9}$$

on obtient la formulation équivalente suivante du problème (METRIC) :

$$\begin{aligned} &\text{minimiser } \rho(\boldsymbol{\omega}) = \max_{l\in E} \max_{\mathbf{d}\in\mathcal{D}_*^\tau} \frac{1}{c_l} y_l(\boldsymbol{\omega}, \mathbf{d}) \\ &\text{avec } \ \boldsymbol{\omega} \in \Omega^M \end{aligned}$$

Cette nouvelle formulation se prête mieux à l'optimisation car, comme nous l'expliquerons dans la section 4.3.2.2, elle permet d'utiliser des algorithmes plus efficaces pour calculer le taux d'utilisation des liens dans le pire cas.

Nous utilisons l'heuristique gloutonne présentée dans l'Algorithme 2 pour résoudre le problème global.

Cette heuristique commence par initialiser la meilleure configuration de poids $\boldsymbol{\omega}^*$ et la configuration courante $\boldsymbol{\omega}$ à $\boldsymbol{\omega}^\tau$. À chaque itération de la boucle d'optimisation, elle sélectionne arbitrairement un lien ℓ parmi les plus congestionnés (ligne 3), et calcule l'incrément minimal de poids Δ_ℓ permettant de dévier au moins un flux (en partie ou entièrement) de ce lien (ligne 4). La configuration courante des poids $\boldsymbol{\omega}$ est alors mise à $\boldsymbol{\omega} + \Delta_\ell \mathbf{e}_\ell$, où $\mathbf{e}_\ell$ est le vecteur de taille M avec la valeur 1 en position ℓ et la valeur 0 partout ailleurs, puis le taux de congestion du réseau avec la configuration $\boldsymbol{\omega}$ est alors évalué (ligne 6). S'il est réduit par rapport à la meilleure configuration $\boldsymbol{\omega}^*$, alors $\boldsymbol{\omega}^*$ est mis à jour (ligne 8). Pour éviter de rester coincé dans un minimum local, l'heuristique autorise l'augmentation du taux de congestion du réseau, pour un nombre

Algorithme 2 Heuristique gloutonne pour l'optimisation robuste des liens.

Require: $\boldsymbol{\omega}^\tau$ et $\mathcal{D}_*^\tau$

1: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega}^\tau$; $\boldsymbol{\omega}^* \leftarrow \boldsymbol{\omega}$
2: **while** $n < N_{max}$ **and** $q < Q_{max}$ **do**
3: Choisir $\ell \in \arg\max_{l \in E} \max_{\mathbf{d} \in \mathcal{D}_*^\tau} \frac{y_l(\boldsymbol{\omega}, \mathbf{d})}{c_l}$
4: $\Delta_\ell \leftarrow \arg\min_{\Delta \geq 1} \left[\sum_k f_{l,k}(\boldsymbol{\omega} + \Delta\, \mathbf{e}_\ell) < \sum_k f_{l,k}(\boldsymbol{\omega}) \right]$
5: $\boldsymbol{\omega} \leftarrow \boldsymbol{\omega} + \Delta_\ell \mathbf{e}_\ell$
6: Calculer $\rho(\boldsymbol{\omega})$
7: **if** $\rho(\boldsymbol{\omega}) \leq \rho(\boldsymbol{\omega}^*)$ **then**
8: $\boldsymbol{\omega}^* \leftarrow \boldsymbol{\omega}$; $q \leftarrow 0$
9: **else**
10: $q \leftarrow q + 1$
11: **end if**
12: $n \leftarrow n + 1$
13: **end while**

limité d'itération $q \leq Q_{max}$. La convergence est atteinte quand $q = Q_{max}$ ou quand le nombre maximum d'itérations est dépassé (ligne 2). Nous décrivons plus en détails les principales étapes de l'heuristique dans les parties suivantes.

4.3.2.1 Incrément minimal de poids Δ_ℓ

L'idée de l'heuristique gloutonne est de dévier du trafic du lien ℓ le plus congestionné au pire cas. Il faut pour cela augmenter le poids de ce lien de la quantité minimale Δ_ℓ tel que au moins un flux k passant par ℓ en soit dévié. Formellement, Δ_ℓ est calculé en utilisant la formule donnée en ligne 4 de la Figure 2.

Nous rappelons que, même si ℓ est le lien avec le taux d'utilisation le plus élevé au pire cas, la quantité Δ_ℓ est indépendante des demandes en trafic. On remarque que s'il existe un flux k tel que $0 < f_{l,k}(\boldsymbol{\omega}) < 1$, alors ce flux est divisé sur plusieurs plus courts chemins. Dans ce cas, l'incrément de poids $\Delta_\ell = 1$ est suffisant pour dévier le flux k du lien ℓ. Pour les autres cas, nous utilisons la technique proposée dans [46]. L'idée est de supprimer le lien ℓ du réseau et d'analyser comment la distance entre la source et la destination augmente pour les flux k tels que $f_{l,k}(\boldsymbol{\omega}) = 1$. En choisissant Δ_ℓ comme l'augmentation minimale de longueur des plus courts chemins, on introduit du partage de charge pour au moins l'un de ces flux, déviant ainsi au moins une partie du flux du lien ℓ. Nous renvoyons le lecteur vers [46] pour de plus amples détails.

4.3.2.2 Évaluation de la charge des liens au pire cas

Une des opérations clés de l'heuristique gloutonne est l'évaluation de la charge des liens au pire cas. Cette opération est nécessaire pour choisir le lien ℓ (ligne 3), et pour calculer le taux de congestion du réseau (ligne 6) après que le poids d'un lien ait été augmenté. Nous commençons par noter qu'il n'est pas nécessaire de recalculer la charge au pire cas pour l'ensemble des liens, mais seulement pour les liens pour lesquels les paramètres $f_{l,k}$ ont été modifiés. Notons l l'un de ces liens.

En remarquant que

$$\sum_n b_n^{e,\tau} - \sum_k d_k^\tau = 0, \tag{4.10}$$

la pire charge du lien l peut s'écrire :

$$\begin{aligned} \max_{\mathbf{d}\in\mathcal{D}_*^\tau} y_l(\boldsymbol{\omega},\mathbf{d}) &= \sum_n b_n^{e,\tau} + \max_{\mathbf{d}\in\mathcal{D}_*^\tau} \sum_k (f_{l,k}(\boldsymbol{\omega})-1)d_k, \\ &= \sum_n b_n^{e,\tau} - \min_{\mathbf{d}\in\mathcal{D}_*^\tau} \sum_k (1-f_{l,k}(\boldsymbol{\omega}))d_k. \end{aligned}$$

Le calcul de $\max_{\mathbf{d}\in\mathcal{D}_*^\tau} y_l(\boldsymbol{\omega},\mathbf{d})$ se réduit alors à résoudre le problème de minimisation suivant :

$$\begin{cases} \min_{\mathbf{d}\in\mathbb{R}_+^K} & \sum_k (1-f_{l,k}(\boldsymbol{\omega}))\, d_k \\ \text{s.t.} & \\ & (1-\gamma)\hat{d}_k \le d_k \le (1+\gamma)\hat{d}_k \quad ,\forall k \\ & \sum_{k:s(k)=n} d_k = b_n^{i,\tau} \quad , \ n\in V \\ & \sum_{k:t(k)=n} d_k = b_n^{e,\tau} \quad , \ n\in V \end{cases}$$

La structure du problème linéaire ci-dessus correspond à celle d'un problème de flux à coût minimal dans un graphe bi-partie. Il peut être résolu très efficacement en utilisant un algorithme dédié [10]. L'avantage d'utiliser le polytope $\mathcal{D}_*^\tau$ au lieu du polytope $\mathcal{D}^\tau$ est précisément ici : cela permet de réduire les temps de calcul en résolvant un problème de flot à coût minimal au lieu d'un problème linéaire plus général. Pour résoudre ce problème, nous utilisons la bibliothèque logicielle LEMON (C++) [3], qui propose des implémentations efficaces d'algorithmes pour la résolution de ce type de problème.

4.3.3 Réduction du nombre de changement de poids

Nous rappelons que pour éviter de rester coincé dans des minimums locaux, certaines modifications de poids peuvent amener à augmenter temporairement le taux de congestion du réseau. Nous regroupons ainsi les changements de poids proposés par

l'heuristique gloutonne en groupes, tel que si tous les changements de poids d'un même groupe sont appliqués, le taux de congestion du réseau est strictement réduit.

Après l'application d'une nouvelle configuration de poids, le protocole de routage intra-domaine requiert un certain temps avant de converger effectivement vers les nouvelles routes. Pour éviter de constamment modifier le routage, il est important de restreindre les modifications à celles qui sont vraiment nécessaires pour améliorer la situation.

Nous utilisons les règles suivantes pour réduire le nombre de changements de poids :

— l'heuristique gloutonne s'arrête si les poids de plus de MAX_CHANGE liens distincts ont été modifiés,

— le dernier groupe de changement réduisant le taux de congestion du réseau de moins d'un certain seuil G_{seuil} est supprimé.

La Figure 4.4 présente une illustration de l'application de ces règles avec $MAX\_CHANGE = 10$. Chaque barre correspond à la production d'un nouveau groupe de changement de métriques. Le nombre cumulé de liens dont les poids ont été changés est indiqué sur la figure pour chaque groupe (on remarque ainsi que ce nombre n'augmente pas entre les étapes 3 et 4 car les nouveaux changements proposés concernent des liens déjà modifiés dans les groupes précédents). Dans cet exemple, l'heuristique propose 6 groupes de changements de poids. Nous ne conservons alors que les quatre premiers car : (a) le groupe 6 excède le nombre de changements de poids autorisé, et (b) le groupe 5 ne permet qu'un gain inférieur au seuil G_{seuil}.

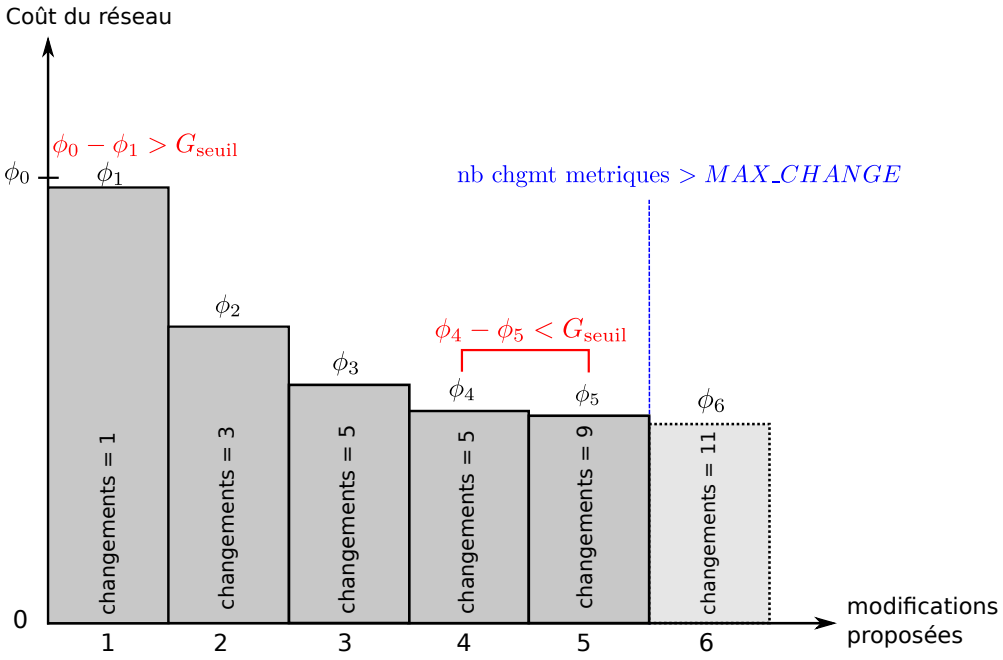


FIGURE 4.4 – Restriction du nombre de changements de poids.

4.4 Résultats

Dans cette section, nous analysons comment les taux de congestion réseau évoluent dans le temps en utilisant l'algorithme en ligne proposé. L'analyse est faite à la fois sur des trafics simulés et des trafics réels.

Tous les résultats présentés ont été obtenus en utilisant les paramètres suivants : $N_{max} = 100$, $Q_{max} = 10$, $G_{seuil} = 2\%$, $MAX\_CHANGE = 10$ et plusieurs valeurs de γ. L'implémentation C++ de l'algorithme a été compilée avec le compilateur GCC, en activant le niveau d'optimisation -03. Toutes les simulations sont exécutées sur une machine utilisant un processeur Intel Core i5-2430M à 2.4 GHz, sur un système d'exploitation Linux disposant de 4 GB de mémoire vive.

4.4.1 Trafics simulés

Les simulations sont exécutées sur 8 topologies réelles (voir Table 4.1). Leurs caractéristiques ont été trouvées dans littérature IEEE (BHVAC, PACBELL, EON, METRO, ARPANET, NSF) et depuis le projet Rocketfuel [108] pour les topologies ABOVENET et VNSL. Pour chaque topologie, le poids initial de chaque lien est pris égal à 1.

TABLE 4.1 – Caractéristiques des topologies : nombre de nœuds (N) et nombre de liens (M).

Topologie	N	M
ABOVENET	19	68
ARPANET	24	100
BHVAC	19	46
EON	19	74
METRO	11	84
NSF	8	20
PACBELL	15	42
VNSL	9	22

Pour chaque réseau, une matrice de trafic aléatoire est générée à l'instant $\tau = 0$. Nous rappelons que le nombre de paires OD vaut $K = N(N-1)$, où N est le nombre de nœuds de la topologie réseau. Chaque minute, la matrice de trafic est mise à jour en ajoutant un bruit blanc gaussien à chaque demande :

$$d_k^t = d_k^{t-1\,minute} + \mathcal{N}(0, \sigma^2) \quad k = 1, \ldots, K. \tag{4.11}$$

La déviation standard de ce bruit est choisie de façon à ce que 99.7% des demandes

varient d'au plus ±8.5% par minute (et nous forçons également ces bornes pour les 0.3% restants). En conséquence, chaque demande peut varier d'au plus ±50% sur un intervalle de 5 minutes :

$$0.5d_k^{\tau-1} \leq d_k^{\tau} \leq 1.5d_k^{\tau-1} \tag{4.12}$$

Malgré sa simplicité, ce modèle permet d'importantes variations du trafic de chaque demande (+-50% sur 5 minutes). De plus, des trafics réels observés sur le réseau ABILENE (voir section 4.4.2 pour des détails) accréditent notre modèle simplifié : les trafics réels mesurés évoluent sur des échelles de temps de plusieurs minutes, et les variations maximales observées sont similaires à celles observées pour nos simulations (max +-50% chaque 5 minutes). Nous insistons sur le fait que notre méthode n'est pas conçue pour prendre en charge d'intenses variations de trafic sur de courtes périodes de temps (inférieures à 5 minutes), qui nécessiteraient de vrais algorithmes de routage dynamique, mais plutôt pour adapter les routes lorsque le trafic fluctue sur de plus grandes échelles de temps, en raison du comportement des utilisateurs (cycle jour/nuit, heures de travail, etc. . .).

Toutes nos analyses sur trafic simulés concernent des périodes de temps de 250 minutes (environ 4 heures).

4.4.1.1 Moyenne temporelle du taux de congestion réseau

Pour analyser les performances obtenues en utilisant l'algorithme en ligne, nous décidons d'étudier la moyenne temporelle du taux de congestion réseau sur la simulation complète. Pour calculer cette valeur le taux de congestion réseau est mesuré chaque minute. La moyenne temporelle du taux de congestion réseau est alors définie comme suit :

$$\bar{\phi} = \frac{1}{n}\sum_{t=1}^{n} \max_{l \in E} \frac{y_l^t}{c_l} = \frac{1}{n}\sum_{t=1}^{n} \max_{l \in E} \left(\frac{\sum_{k=1}^{K} f_{l,k}(\boldsymbol{\omega}^t) d_k^t}{c_l} \right) \tag{4.13}$$

avec dans notre cas, $n = 250$.

Trois configurations de poids sont considérées. Les deux premières sont communément utilisées dans les réseaux réels , et correspondent à des poids unitaires ($\boldsymbol{\omega}_l = 1, \forall l \in E$) et des poids CISCO ($\boldsymbol{\omega}_l = 10^8/c_l, \forall l \in E$), respectivement. La troisième configuration considérée est hypothétique et utilise des poids optimisés. Pour obtenir ces poids optimisés, nous supposons connaître les demandes en trafic futures pour la totalité de la période de simulation. Nous calculons alors la matrice de trafic moyenne sur les 250 minutes de simulations, et utilisons ensuite l'algorithme de recherche locale proposé dans [24] pour optimiser les poids en tenant compte de cette matrice moyenne. Enfin, ces

poids optimisés sont utilisés pour définir une configuration statique optimisée pour la période de simulation (i.e. ces poids ne sont pas changés durant la simulation).

Ces trois configurations statiques de poids sont comparées avec l'algorithme en ligne que nous proposons. Comme le paramètre γ (défini dans la section 4.3.2) affecte l'évaluation des charges des liens au pire cas, il peut modifier les décisions de routage de l'algorithme. Pour mieux observer l'influence de ce paramètre, toutes les simulations sont effectuées avec trois valeurs $\gamma \in \{0, 0.25, 0.4\}$. On note que la configuration $\gamma = 0$ revient à désactiver complètement l'incertitude sur la demande et utilise directement la matrice tomogravité. Les résultats sont présentés sur la Table 4.2.

TABLE 4.2 – Moyenne des taux de congestion réseau (%) sur les 250 minutes de simulation.

Topologie	**Poids statiques**			**Algorithme en ligne**		
	Unitaire	**CISCO**	**OPT**	$\boldsymbol{\gamma = 0}$	$\boldsymbol{\gamma = 0.25}$	$\boldsymbol{\gamma = 0.4}$
ABOVENET	81.00	101.20	55.69	81.00	81.00	81.00
ARPANET	81.57	96.35	75.84	81.57	81.57	81.57
BHVAC	81.62	75.13	67.30	81.62	64.80	65.62
EON	91.21	103.49	54.86	91.21	50.95	47.63
METRO	63.08	47.83	44.13	63.08	37.84	37.93
NSF	43.78	43.77	41.52	43.78	42.16	43.52
PACBELL	137.97	40.04	37.56	46.15	38.14	39.47
VNSL	59.28	59.28	58.43	59.28	59.28	59.28
Moyenne	79.94	70.89	54.42	68.46	56.97	57.00

Plusieurs commentaires sont nécessaires. Premièrement, les résultats montrent que des bénéfices significatifs peuvent être attendus d'une optimisation des poids des liens. En effet, l'algorithme en ligne permet d'obtenir des congestions réseau clairement meilleures que les configurations statiques Unitaires ou CISCO, et ce pour toutes les valeurs de γ différentes de 0, à l'exception des topologies ABOVENET, ARPANET et VNSL, pour lesquelles aucune modification acceptable n'a été proposée sur ces instances.

Nous notons également la proximité entre les résultats obtenus avec l'algorithme en ligne et ceux obtenus avec la configuration statique optimisée (OPT). Nous insistons sur le fait que cette configuration statique optimisée est purement théorique car elle nécessite la connaissance des trafics futurs pour l'ensemble de la période étudiée, ce qui est impossible dans la réalité. De plus, la durée de la simulation considérée est relativement courte (250 min), ce qui signifie que les demandes en trafic peuvent rester “proches" de leurs valeurs moyennes sur la période. Plus la période de temps considérée pour calcu-

ler cette configuration optimisée s'allongera, plus la probabilité que la matrice de trafic réelle dévie significativement de la matrice de trafic moyenne augmentera, et moins la configuration statique optimisée restera intéressante par rapport à une reconfiguration dynamique.

Enfin, nous notons les bénéfices tirés de l'utilisation d'une part d'incertitude dans l'estimation des matrices de trafic. Dans les faits pour les instances testées, avec $\gamma = 0$ (considérant donc uniquement la matrice tomogravité, sans aucune incertitude), les taux de congestion sont identiques à ceux fournis par la configuration statique unitaire (à l'exception de la topologie PACBELL), car aucune modification intéressante n'est trouvée par l'algorithme. En pratique pour ces cas, soit l'algorithme ne trouve pas de modifications acceptables (offrant un gain suffisant supérieur à G_{seuil}), soit le nombre de modifications est trop important et dépasse le seuil MAX_CHANGE dès le premier groupe de modifications. On voit que l'utilisation de l'ensemble d'incertitude permet d'améliorer les choses, et permet à l'algorithme de proposer et d'appliquer des modifications pertinentes.

En raison de la nature heuristique de l'algorithme, mais aussi de l'erreur d'estimation non prévisible, il est difficile de choisir et de conseiller la "meilleure" valeur de γ, car les résultats sont très proches entre $\gamma = 0.25$ et $\gamma = 0.4$. Les résultats peuvent de plus fluctuer en fonction des instances considérées. Dans ce qui suit, en nous basant sur nos résultats de simulation, nous décidons d'utiliser la valeur $\gamma = 0.25$, qui semble fournir un bon compromis.

Les nombres totaux de poids modifiés sont exposés dans la Table 4.3. Ils restent assez limités. La Table 4.4 montre les nombres d'instants où une reconfiguration a été opérée par l'algorithme dynamique. Nous voyons que les changements ne sont appliqués que sur un petit nombre d'instants, évitant ainsi de changer continuellement les routes sur le réseau. On note que la topologie BHVAC est sujette à plus d'instants de modifications que les autres topologies.

4.4.1.2 Évolution temporelle des taux de congestion réseau

Dans cette section, nous nous intéressons à l'évolution temporelle du taux de congestion réseau. Nous voulons aussi étudier quand les changements de poids sont appliqués, et leurs conséquences sur le taux de congestion réseau. Dans cette perspective, nous décidons de représenter graphiquement les taux de congestion, et les modifications appliquées durant la période de simulation.

Pour évaluer les gains obtenus par l'utilisation d'un routage dynamique, nous comparons avec la configuration statique où tous les poids sont égaux à 1 (configuration Unitaire). Pour estimer la perte de performance due à l'incertitude d'estimation sur les trafics, nous comparons aussi avec les résultats obtenus avec l'heuristique gloutonne utilisant les vrais trafics courant (et non plus une estimation sur les 5 dernières minutes).

TABLE 4.3 – Nombre total de changement de poids sur les 250 minutes de simulation.

Topologie	$\gamma = 0$	$\gamma = 0.25$	$\gamma = 0.4$
ABOVENET	0	0	0
ARPANET	0	0	0
BHVAC	0	34	49
EON	0	28	42
METRO	0	25	28
NSF	0	6	12
PACBELL	1	9	9
VNSL	0	0	0

TABLE 4.4 – Nombre d'instants où les poids ont été modifiés, sur les 250 minutes de simulation.

Topologie	$\gamma = 0$	$\gamma = 0.25$	$\gamma = 0.4$
ABOVENET	0	0	0
ARPANET	0	0	0
BHVAC	0	16	17
EON	0	4	6
METRO	0	5	6
NSF	0	2	3
PACBELL	1	2	1
VNSL	0	0	0

Enfin, nous souhaitons observer l'écart par rapport à la congestion réseau qui pourrait être obtenue avec une configuration optimisée à chaque instant en connaissant exactement les trafics. Comme il n'existe pas d'algorithme d'optimisation exact pour résoudre le problème d'optimisation des poids de liens, nous calculons plutôt une borne inférieure sur le taux de congestion réseau en résolvant le problème de routage multichemins suivant :

$$\min\ z \qquad \text{(MULTICHEMINS)}$$

avec

$$\sum_k f_{l,k}\, d_k^{\tau+1} \leq c_l\, z,\ \forall l \in E, \tag{4.14}$$

$$\sum_{l\in\delta^+(n)} f_{l,k} - \sum_{l\in\delta^-(n)} f_{l,k} = h_k^n,\ \forall k, \forall n \in V, \tag{4.15}$$

$$0 \leq f_{l,k} \leq 1,\ \forall l \in E,\ \forall k \tag{4.16}$$

où $\delta^+(n)$ (resp. $\delta^-(n)$) représente l'ensemble des liens entrants (resp. sortants) du nœud n, avec h_k^n égal à 1 si $v = s(k)$, -1 si $v = t(k)$ et 0 autrement. Les inégalités (4.16) assurent qu'il n'existe aucune restriction sur le schéma de routage. Nous supposons ici que la future demande en trafic $d_k^{\tau+1}$ est connue à l'instant τ. Nous soulignons que cette borne inférieure est potentiellement bien meilleure que le meilleur taux de congestion atteignable avec l'utilisation du schéma de routage OSPF.

Les Figures 4.5, 4.6 et 4.7 présentent les résultats obtenus pour les topologies BHVAC, EON et METRO, respectivement (des résultats similaires sont obtenus pour les autres topologies). Ces figures montrent l'évolution du taux de congestion réseau. L'axe des abscisses représente le temps τ. Deux axes verticaux sont présents : l'axe de gauche est le taux de congestion réseau mesuré, tandis que celui de droite indique le nombre de changements de poids opérés par l'algorithme en ligne. Quatre courbes sont présentes sur chaque graphique :

- Lowerbound : la borne inférieure obtenue en résolvant le problème (MULTICHEMINS),
- UnitMetrics : le taux de congestion réseau obtenu avec la configuration statique constituée de poids unitaires,
- KnowMatrixOptim : le taux de congestion réseau obtenu avec l'heuristique gloutonne, si la matrice de trafic était connue,
- EstimatedMatrixOptim : le taux de congestion réseau obtenu avec l'algorithme en ligne que nous proposons.

Les barres verticales représentent le nombre de changements de poids opérés par l'algorithme en ligne proposé. L'absence de barre verticale signifie que l'algorithme ne modifie pas le routage.

Nous remarquons tout d'abord que l'utilisation d'un schéma de routage dynamique permet de réduire significativement le taux de congestion réseau par rapport à une configuration statique tel que les poids unitaires.

Nous observons aussi que pour toutes les topologies, il semble que l'incertitude sur les demandes estimées n'entraîne pas de pertes significatives de performances. En pratique,

l'estimation de matrice de trafic semble suffisamment précise pour opérer une optimisation intéressante. L'algorithme d'optimisation robuste se comporte même parfois mieux que si la matrice était connue : ce phénomène s'explique par la nature heuristique de l'algorithme et par le fait que l'on optimise les routes futures en utilisant les trafics courants.

On note que pour les topologies BHVAC et EON, le taux de congestion réseau obtenu avec l'algorithme en-ligne est souvent assez proche de la borne inférieure.

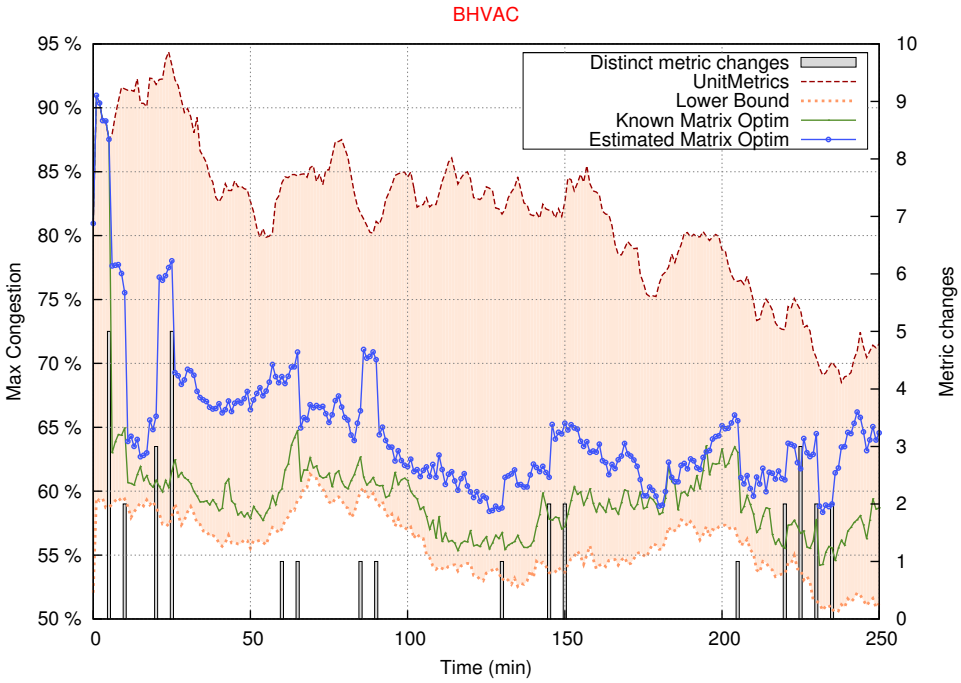


FIGURE 4.5 – Taux de congestion réseau pour la topologie BHVAC.

4.4.1.3 Temps d'exécution

Le temps d'exécution de l'algorithme est critique dans cette étude. Les pires temps d'exécution sont présentés dans la Table 4.5. Pour tous les réseaux, les décisions de routage sont prises en quelques fractions de secondes, ce qui est clairement compatible avec un fonctionnement en ligne. Une simulation sur deux plus grosses topologies, que nous appelons BRITE1(50 nœuds, 200 liens) et BRITE2 (100 nœuds, 400 liens), générées avec l'outil BRITE [1], nous a montré que les temps d'exécution peuvent augmenter rapidement avec la taille du réseau. Ainsi le pire temps d'exécution est alors égal à 4.6 secondes pour BRITE1 et 24 secondes pour BRITE2. Ce comportement est dû à la taille du problème considéré, qui s'accroît très rapidement avec la taille du réseau (par exemple, les fractions de routage $f_{l,k}$ sont composées de $N^2 \cdot M$ valeurs qui doivent être

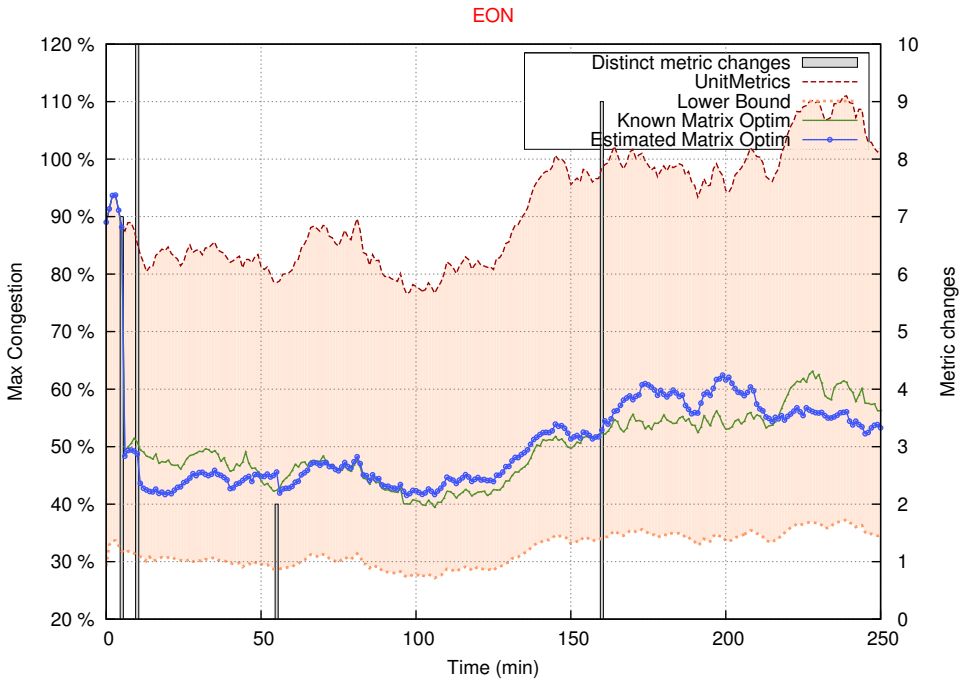


FIGURE 4.6 – Taux de congestion réseau pour la topologie EON.

calculées et utilisées). Ainsi, sur la topologie BRITE2, l'estimation de la matrice de trafic nécessite environ 4.6 secondes, et les calculs répétés du taux de congestion au pire cas (cf section 4.3.2.2) représentent un temps total d'environ 5.5 secondes. Le temps restant est passé à calculer les routes et en particulier les fractions de routage $f_{l,k}$. Malheureusement, les fournisseurs d'accès Internet considèrent leurs topologies comme une donnée sensible, et elles ne sont donc pas connues. Les données mises à disposition du public (voir Table 4.1) ainsi que les informations disponibles depuis le réseau ABILENE (cf Section 4.4.2) suggèrent tout de même que les réseaux de cœur peuvent comporter moins de nœuds que la topologie BRITE2.

4.4.2 Trafics réels

Nous présentons ici les résultats obtenus avec des informations de trafics réels enregistrées en 2004 sur le réseau ABILENE (12 nœuds et 30 liens)[120]. Malheureusement, nous n'avons pas pu obtenir des traces de trafic plus récentes, car elles ne sont pas publiquement disponibles. Une matrice de trafic plus récente présenterait probablement des volumes de données plus importants, même s'il est très difficile d'imaginer la forme des trafics.

Les données sur les trafics transitant sur le réseau ABILENE ont été collectées toutes les 5 minutes pendant 24 semaines (6 mois). Nous choisissons de mener nos simulations

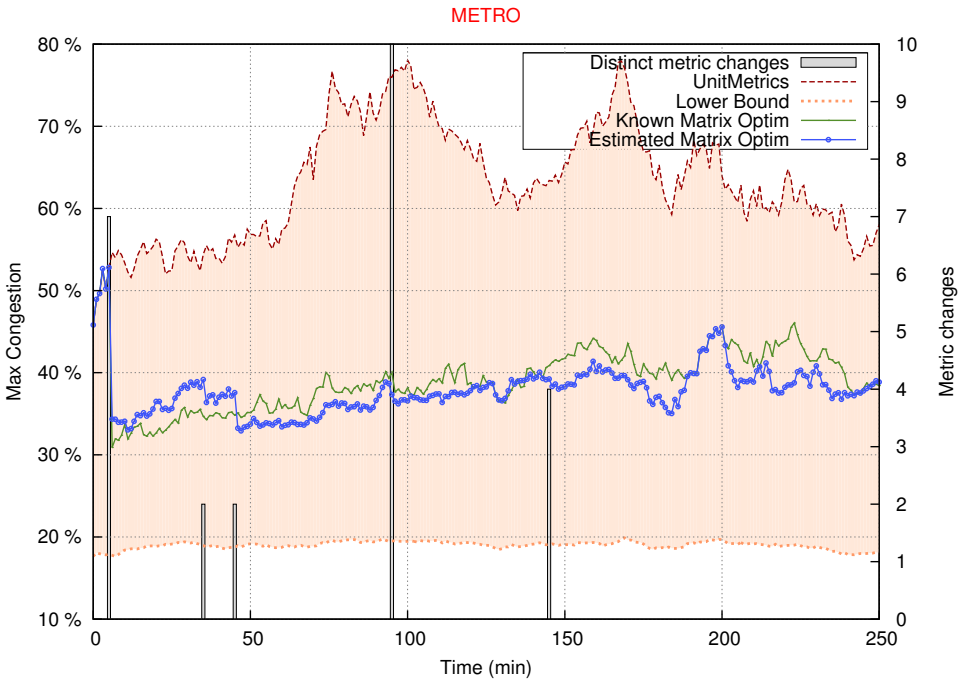


FIGURE 4.7 – Taux de congestion réseau pour la topologie METRO.

TABLE 4.5 – Pires temps d'exécution (s).

Topologie	**Temps**
ABOVENET	0.15
ARPANET	0.56
BHVAC	0.17
EON	0.21
METRO	0.07
NSF	0.04
PACBELL	0.10
VNSL	0.04

sur les données d'une semaine complète de cet enregistrement. Nous utilisons 3 critères pour le choix de la semaine. Premièrement, nous voulons une variabilité des trafics journalière bien visible. Deuxièmement, nous souhaitons aussi pouvoir observer d'importantes variations de trafics sur de courtes périodes temporelles. Enfin, nous avons choisi une semaine pour laquelle l'écart entre la moyenne des taux de congestion des liens et le taux de congestion réseau (max des taux d'utilisations des liens) est suffisamment

important, ce qui indique qu'une optimisation est potentiellement envisageable. La quatrième semaine de l'enregistrement respecte l'ensemble de ces critères, et c'est pourquoi nous l'avons choisie pour mener à bien nos simulations.

L'évolution du taux de congestion réseau est présentée sur la Figure 4.8, avec des poids statiques de type unitaires. Le taux moyen de congestion des liens y est aussi représenté. On remarque que les variations du taux moyen de congestion sont moindres que celles du taux de congestion réseau, ce qui indique un vrai potentiel d'optimisation pour une meilleure répartition de la charge sur le réseau.

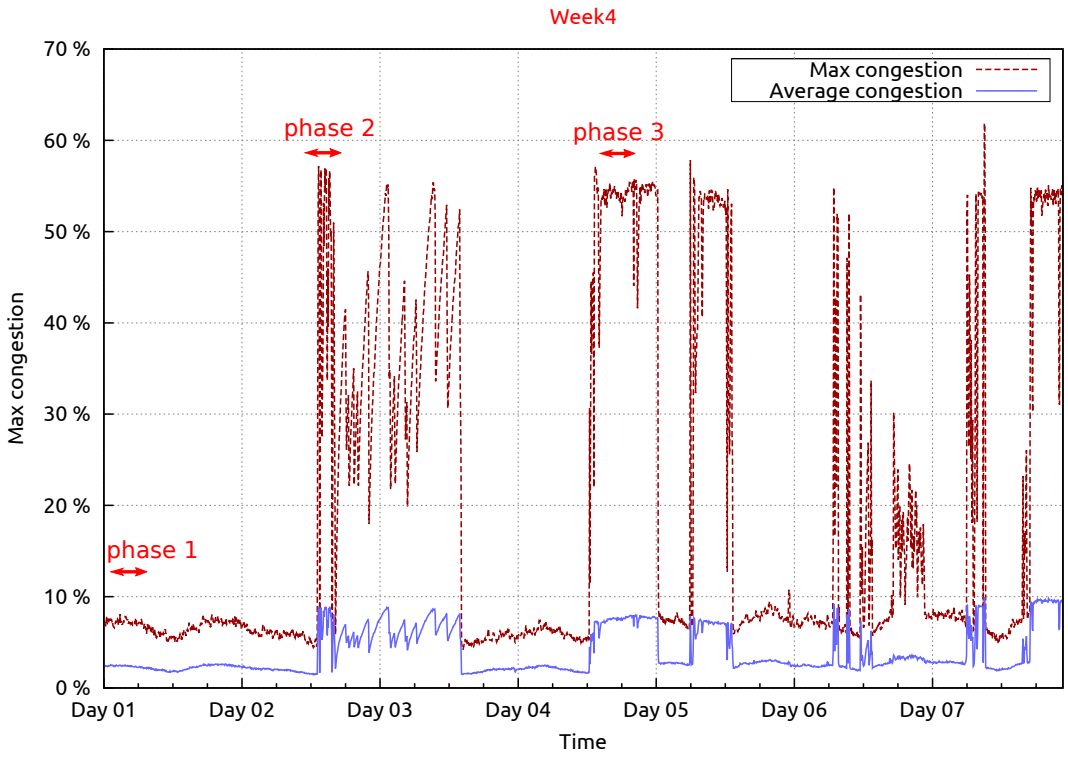


FIGURE 4.8 – Taux de congestion pour la 4ème semaine sur ABILENE, avec poids statiques unitaires.

Nous analysons ici plus en détails l'évolution des trafics pendant trois phases d'une heure dans cette semaine (voir Figure 4.8) : la phase 1 correspond à une période de faible activité ($\approx$ 10%), la phase 2 à une période de montée en charge avec d'importantes variations de trafic(voir Figure 4.9), et la phase 3 est une période de plus forte congestion($\approx$ 55%). Pour les trois phases, nous calculons la demande en trafic moyenne pour chaque couple origine-destination pour la période d'une heure. Nous les classons ensuite par ordre décroissant de volume de demandes. Nous conservons alors unique-

ment les premières demandes représentant au moins 90% du trafic total (flux les plus importants pour l'ingénierie du trafic). La Table 4.6 indique des données statistiques concernant les variations relatives (en valeur absolue) de ces flux dont la demande est importante.

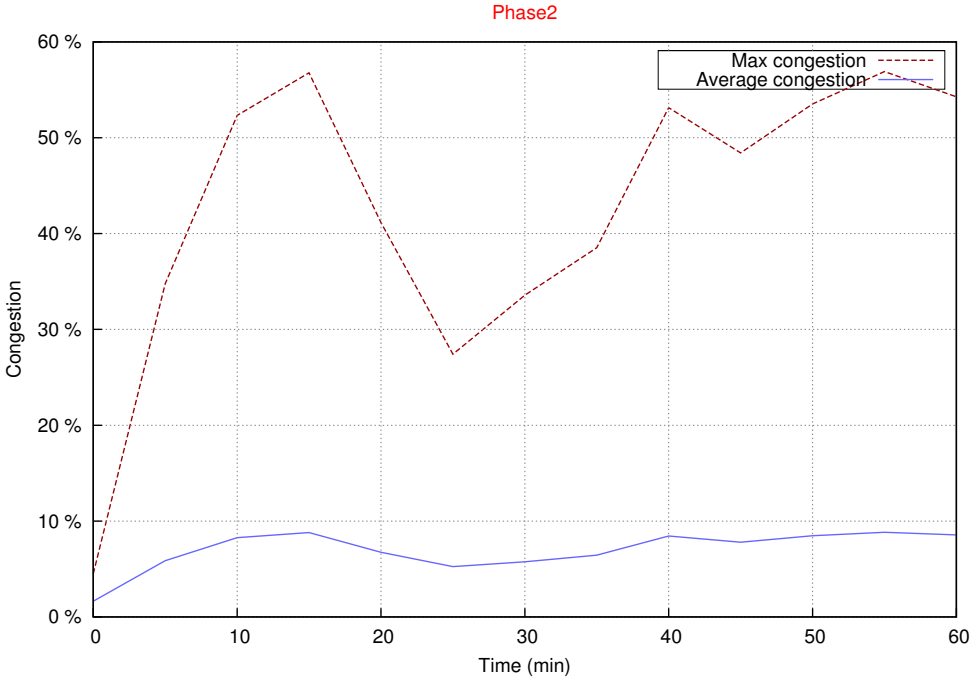


FIGURE 4.9 – Zoom sur la phase 2 du trafic ABILENE étudié.

On remarque que 98% des flux subissent des variations relatives inférieures à 43%, ce qui indique que la contrainte (4.12) que nous utilisons pour nos trafics simulés est bien satisfaite pour la majorité des demandes k. La valeur importante observée pour le 100ème centile de la phase 2 est causée par des trafics quasi nuls à un instant, et devenus non négligeables à l'instant suivant.

TABLE 4.6 – Variations maximum des trafics (%) pour les demandes les plus importantes.

Centile (%)	**Phase 1**	**Phase 2**	**Phase 3**
100	83	142395	87
99	41	74	48
98	33	43	35
95	21	31	26
90	16	25	18

La même étude est effectuée pour les trafics entrants/sortants des nœuds du réseau ($b_n^{i;\tau}$ et $b_n^{e;\tau}$). Les résultats obtenus pour les trafics entrants et sortants sont similaires, et la Table 4.7 montre des informations statistiques pour les trafics entrants. On observe que 90% des trafics entrants subissent des variations relatives inférieures à 19% pour la phase 2 et inférieures à 12 % pour les phases 1 et 3. Bien que ces variations ne soit pas complètement négligeables, les résultats d'optimisation obtenus montre que cela n'affecte pas la qualité d'optimisation obtenue.

TABLE 4.7 – Variations maximales (%) des volumes de trafic sortants pour l'ensemble des nœuds du réseau ABILENE.

Centile (%)	**Phase 1**	**Phase 2**	**Phase 3**
100	49	1062	36
95	15	28	14
90	10	19	12

Les résultats d'optimisation obtenus pour la topologie ABILENE, avec $\gamma = 0.25$, avec la 4ème semaine de trafics réels, sont présentés sur la Figure 4.10.

L'algorithme d'optimisation dynamique proposé permet une réduction significative de la congestion réseau quand le trafic devient important : le taux de congestion réseau maximum est réduit à environ 45% là où il atteint environ 55% pour les poids statiques unitaires.

En utilisant des poids statiques optimisés (obtenus à partir d'une matrice de trafic moyenne sur la semaine), le taux de congestion réseau moyen pour l'intégralité de la quatrième semaine est présenté dans la Table 4.8. Comme attendu, l'algorithme en ligne fournit des résultats légèrement meilleurs que la configuration statique optimisée. De plus, pour chaque configuration, quand le trafic est important, le taux de congestion réseau est d'environ 45%. Ce faible écart entre les deux types de configuration est un bon résultat si l'on garde à l'esprit que la configuration statique optimisée est purement hypothétique, car l'on ne connaît pas les trafics futurs dans les cas réels.

TABLE 4.8 – Taux de congestion réseau moyen (%) pour l'intégralité de la semaine 4 du réseau ABILENE.

Configuration	**Taux de congestion moyen**
Poids unitaires	19.37 %
Poids optimisés statiques	17.84 %
Algorithme en ligne (γ=0.25)	16.18 %
Algorithme en ligne (γ=0.4)	16.42 %

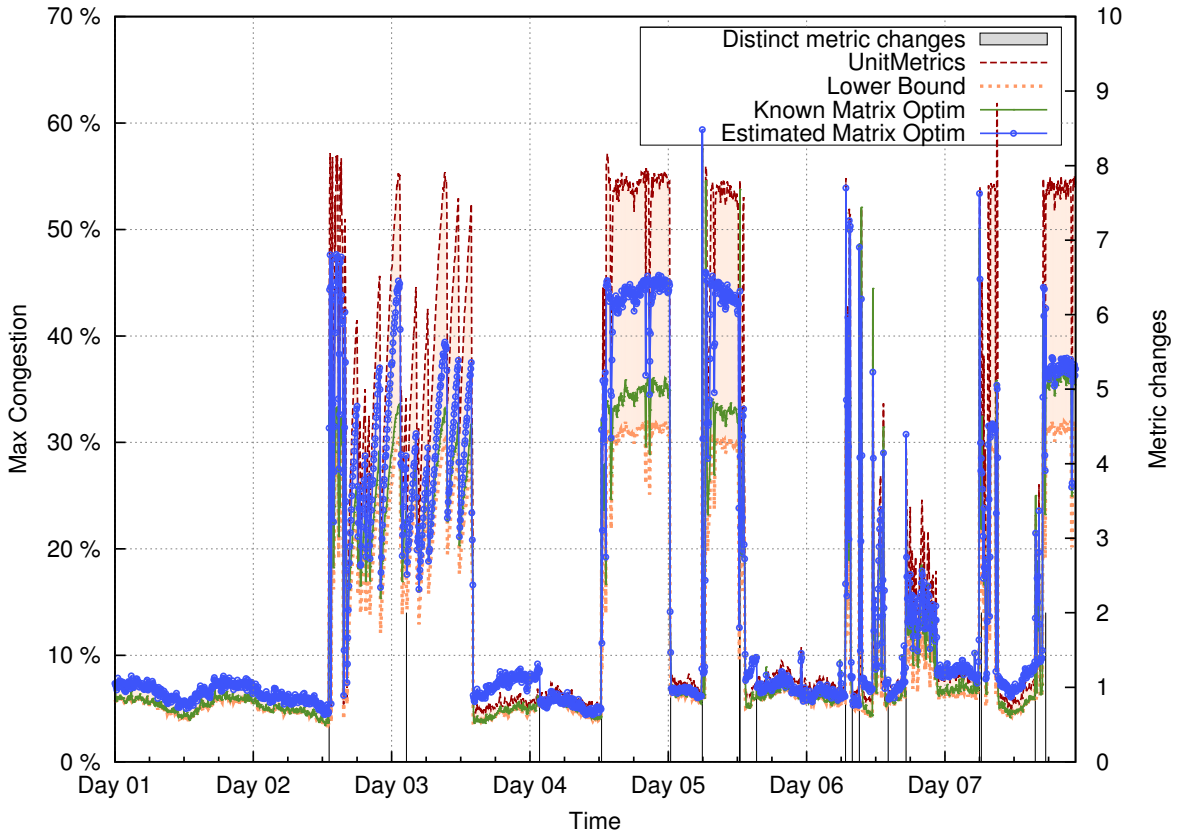


FIGURE 4.10 – Taux de congestion réseau optimisé sur la 4ème semaine du réseau ABILENE.

Les temps d'exécution restent inférieurs à 0.07 secondes, ce qui est clairement compatible avec un fonctionnement en ligne. Le nombre total de modifications reste aussi assez limité pour l'intégralité de la semaine. Pour $\gamma = 0.25$, des modifications ont été appliquées à seulement 17 instants pour l'ensemble de la semaine (qui se compose de 2000 instants d'exécution de l'algorithme), et seulement 27 poids ont été modifiés pour toute la semaine.

4.4.3 Panne de liens

Dans cette partie, nous étudions les résultats obtenus avec l'algorithme proposé lorsqu'une panne de lien apparaît sur le réseau. Plus précisément, nous comparons le taux de congestion réseau obtenu avec les poids unitaires avec ceux fournis par l'algorithme en ligne, lorsqu'un lien tombe en panne à un instant arbitraire T_{down}, avant de réapparaître à l'instant $T_{up} > T_{down}$. Nous utilisons le même protocole de simulation que

dans la section 4.4.1. Le même lien est rompu pour les différentes configurations : il s'agit du lien ayant le taux d'utilisation le plus important à l'instant de la panne. Trois exemples de résultats sont présentés sur les Figures 4.11, 4.12 et 4.13, tous obtenus avec $T_{down} = 97$ème minute, $T_{up} = 157$ème minute et $\gamma = 0.25$. Les mêmes types de résultats sont obtenus pour les autres topologies.

Comme indiqué dans la section 4.3.1, le problème d'estimation de la matrice de trafic peut devenir insoluble si un lien tombe en panne ou réapparaît pendant la période de mesure. Dans ces deux cas, l'algorithme reste en sommeil jusqu'à ce que la prochaine mesure SNMP "valide" (sans modification de topologie) soit disponible. Pour les $T_{down} =$ 97ème minute et $T_{up} = 157$ème minute, cela signifie que l'algorithme ne propose pas de modifications à la 100ème minute et à la 160ème minute.

Les taux de congestion réseau obtenus sont la plupart du temps inférieurs à ceux de la configuration statique unitaire. Nous observons qu'après chaque événement, des modifications sont appliquées, réduisant le taux de congestion réseau d'une façon efficace. Nous remarquons aussi que pour chaque événement, le taux de congestion réseau augmente moins avec l'algorithme en ligne, tout en conservant un nombre de modifications limité. Il semble qu'une bonne répartition des trafics (comme opéré avec l'algorithme) avant une panne de lien permette de réduire l'impact de l'événement sur la congestion du réseau.

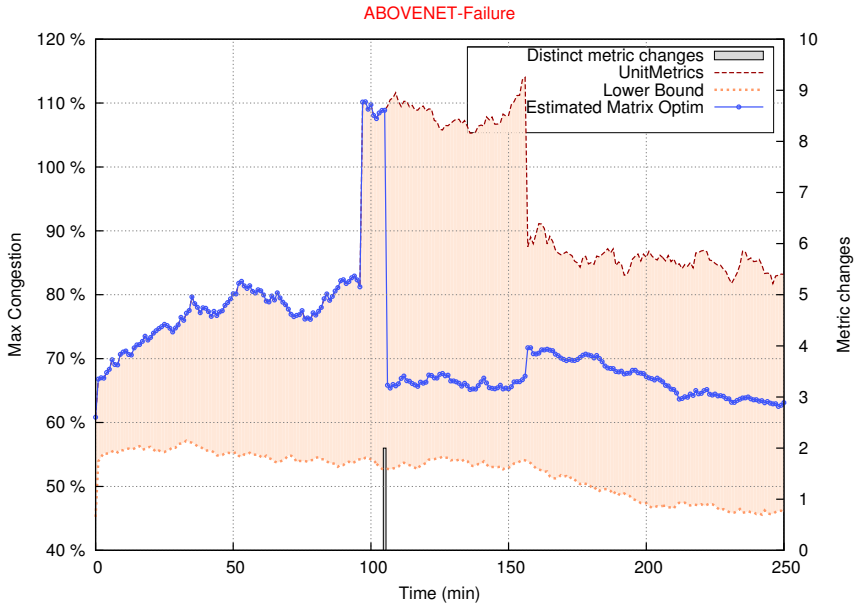


FIGURE 4.11 – Taux de congestion pour la topologie ABOVENET en cas de panne de lien.

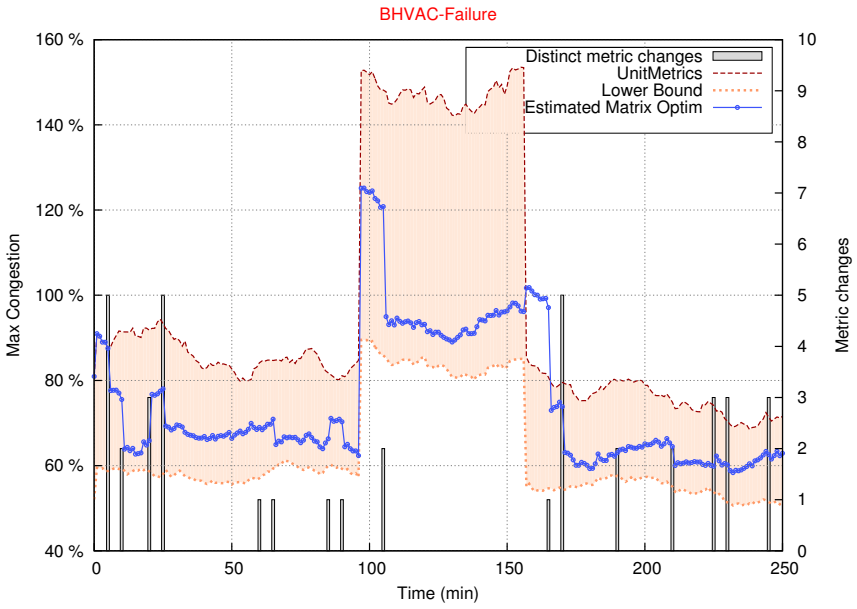


FIGURE 4.12 – Taux de congestion pour la topologie BHVAC en cas de panne de lien.

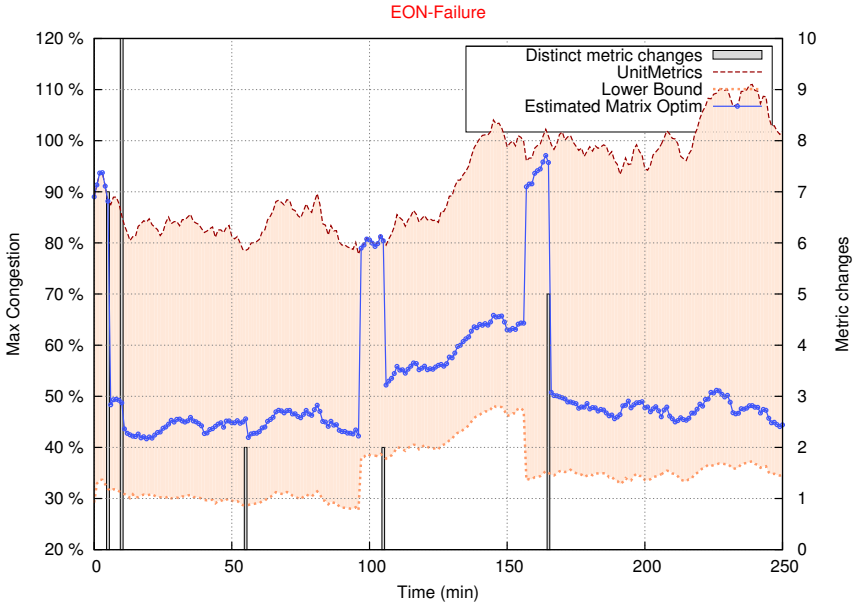


FIGURE 4.13 – Taux de congestion pour la topologie EON en cas de panne de lien.

4.5 Conclusion

L'idée clé de l'approche proposée consiste à dévier du trafic des liens les plus congestionnés. Pour évaluer dans quelle mesure une déviation peut être intéressante, nous avons besoin d'information sur les trafics. Cette information est obtenue via une estimation simple de la matrice de trafic courante, à partir de mesures données par le protocole SNMP. L'optimisation robuste est utilisée pour compenser les erreurs d'estimation du trafic. Les résultats obtenus avec l'algorithme glouton proposé, sur des trafics simulés et des trafics réels montrent qu'une réduction importante du taux de congestion réseau peut être obtenue par rapport à une configuration statique des poids, même si les trafics subissent d'importantes variations temporelles. Pour les trafics sur le réseau ABILENE, le taux maximum de congestion réseau peut être réduit à environ 45% avec l'algorithme en ligne , là où il atteint 55% avec la configuration unitaire des poids. Cela montre aussi que l'algorithme dynamique fournit une configuration plus robuste, en apportant la possibilité de réagir aux variations de trafic. Les temps d'exécution sont compatibles avec un fonctionnement en ligne, tant que l'on ne considère pas des topologies de tailles très importantes. Une autre conclusion intéressante est que, en répartissant mieux les trafics entre les liens du réseau, l'algorithme en ligne proposé aide à réduire l'impact des pannes sur le taux de congestion au moment où l'événement arrive. Une perspective future intéressante consisterait à tester l'implémentation de l'algorithme sur un vrai réseau pour démontrer son utilité et sa faisabilité réelle.

5

Optimisation du placement des LSP dans les réseaux MPLS

5.1 Introduction

Dans ce chapitre, nous nous intéressons à l'optimisation des cœurs de réseau reposant sur la technologie MPLS. Comme évoqué dans le chapitre 1, la technologie MPLS a des avantages certains en termes d'ingénierie de trafic par rapport à des protocoles de routage classiques comme OSPF. En particulier, elle permet de gérer finement l'utilisation des ressources disponibles en assignant un chemin explicite à chacun des LSP, qui sont alors appelés ER-LSP (Explicitly Routed LSP). Un exemple de placement explicite de LSP est présenté sur la Figure 5.1, où 3 LSP ayant la même source et la même destination transitent par des chemins distincts. Avec l'utilisation de LSP explicitement routés, il devient possible d'envisager adapter les chemins utilisés en fonction des conditions de trafic pour optimiser dynamiquement l'utilisation des ressources. C'est cette possibilité que nous exploiterons dans ce chapitre.

Chaque LSP devant être routé sur un et un seul chemin, le problème du placement optimal des LSP est un problème de routage mono-chemin qui peut se formuler avec des variables de décision binaires. Il est de tradition en optimisation de réseaux de chercher à minimiser le taux de congestion du réseau, c'est-à-dire le taux maximal d'utilisation des liens du réseau. Dans ce cas, le placement optimal des LSP peut s'écrire comme un problème linéaire en nombres entiers. D'autres critères de performance peuvent cependant être pertinents, et notamment ceux représentant la qualité de service des flux dans

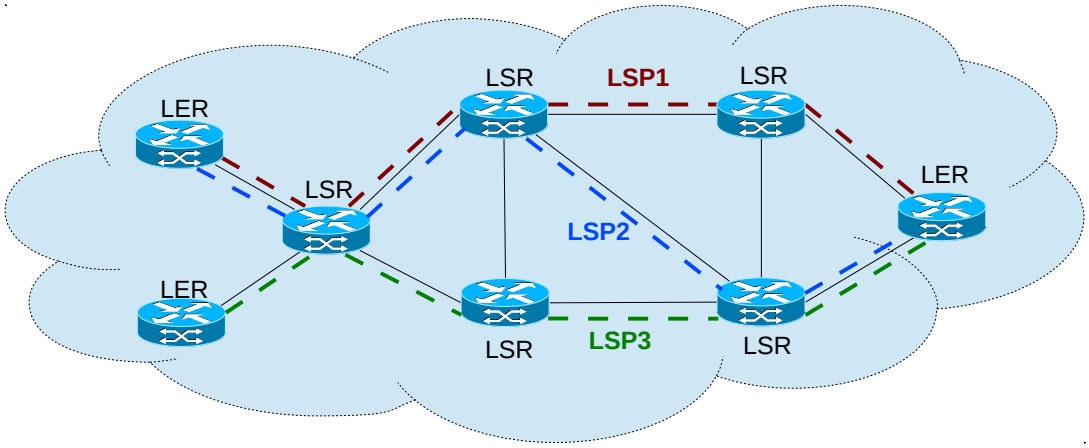


FIGURE 5.1 – Exemple de placement de 3 LSP dans un réseau MPLS.

le réseau. On peut ainsi chercher à minimiser le délai moyen des paquets dans le réseau. C'est par exemple le choix qui a été fait dans [52] où les auteurs cherchent à minimiser la violation d'une contrainte sur le délai des paquets, en supposant que certains trafics sont prioritaires, les autres se partageant le reste de la bande passante en fonction de priorités relatives. Le délai moyen des paquets s'exprimant à l'aide de formules de files d'attente qui sont non linéaires, le problème d'optimisation à résoudre devient alors un problème non linéaire en variables binaires.

Pour prendre en compte ce type de critère, nous étudions dans ce chapitre le problème du placement optimal des LSP en supposant une fonction objectif additive et non linéaire. Ce problème appartient à la classe des problèmes mathématiques non linéaires mettant en jeu des variables entières (binaires, dans notre cas), problèmes qui sont connus pour être très difficiles à résoudre, aussi bien d'un point de vue théorique que pratique. Même les cas les plus simples avec des variables binaires, une fonction coût quadratique et des contraintes d'égalité sont connus pour être NP-difficiles [27]. En pratique, on doit donc souvent se contenter d'une résolution approchée.

Plusieurs approches ont été proposées pour la résolution approchée des problèmes d'optimisation non linéaires en nombres entiers. Certaines transforment le problème discret en un problème continu (voir par exemple [90]). Malgré leurs qualités, ces techniques rencontrent des difficultés de mise à l'échelle vis-à-vis de la taille des problèmes considérés. Une alternative récente est la technique nommée *Global Smoothing Algorithm* [82], qui semble être plus tolérante aux des problèmes de passage à l'échelle, tout en fournissant des approximations intéressantes (voir Section 5.4.1 pour les détails).

Des heuristiques et méta-heuristiques ont aussi été utilisées pour résoudre les pro-

blèmes non linéaires en nombres entiers. Parmi ces dernières, les techniques d'optimisation inspirées du fonctionnement des colonies de fourmis sont connues pour fournir des solutions de bonne qualité dans divers problèmes de routage [52], [103]. Dans ce chapitre, nous utilisons la méthode heuristique proposée dans [52] à des fins de comparaison (voir Section 5.4.2 pour les détails).

Nous proposons ici une heuristique, inspirée de la théorie des jeux [51], pour résoudre les problèmes non-linéaires de routage mono-chemin. L'idée est de supposer que les flux individuels peuvent choisir indépendamment leur chemin afin de minimiser leur propre fonction coût. On note qu'un algorithme basé sur la même idée a été proposé dans [11], pour l'ordonnancement de tâches strictement périodiques. Nous démontrons la convergence de l'heuristique proposée et obtenons des garanties de performance dans quelques cas. Comme le montrent les résultats expérimentaux, le principal intérêt de l'algorithme proposé est qu'il s'avère beaucoup plus rapide que les méthodes évoquées précédemment, tout en fournissant un routage mono-chemin de bonne qualité, avec une erreur relative modeste par rapport aux solutions optimales.

Nous commençons par formuler au paragraphe 5.2 le problème du placement optimal des LSP dans les réseaux MPLS. Nous présentons ensuite au paragraphe 5.3 l'heuristique proposée et établissons certaines de ses propriétés. Nous décrivons les deux autres méthodes utilisées à des fins de comparaison au paragraphe 5.4. Les résultats expérimentaux présentés au paragraphe 5.5 montrent que l'algorithme proposé permet d'obtenir des solutions de bonne qualité avec des temps d'exécution significativement plus petits que les autres méthodes. Ces résultats étant encourageants, nous terminons ce chapitre en évaluant au paragraphe 5.6 le comportement de l'algorithme proposé dans le cas d'une optimisation dynamique lorsque la demande en trafic évolue au cours du temps.

5.2 Problème de routage mono-chemin des LSP

On considère un réseau représenté par un graphe orienté noté $G = (V, E)$. A chaque lien $e \in E$ est associée une fonction de latence $\ell_e : \mathbb{R}_+ \to \mathbb{R}_+$ croissante. Pour chaque ensemble $\pi \subset E$, on définit une constante δ_π^e égale à 1 si $e \in \pi$, et 0 sinon.

On considère un ensemble $\mathcal{K} = \{1, 2, \ldots, K\}$ de couples Origine-Destination (OD). On note s_k et t_k l'origine et la destination du couple OD k, et $\lambda_k \in \mathbb{N}$ sa demande en trafic. Chaque demande en trafic doit être routée dans le réseau sur un unique chemin. Soit Π_k l'ensemble des chemins disponibles pour router le trafic entre s_k et t_k. Une stratégie de routage est définie par le vecteur $\boldsymbol{\pi} = (\pi_k)_{k\in\mathcal{K}} \in \Pi_1 \times \Pi_2 \times \ldots \times \Pi_K$, où π_k est le chemin assigné à la demande en trafic k. L'objectif est de trouver une stratégie de routage minimisant le coût du réseau $F(\boldsymbol{\pi}) = \sum_{e\in E} y_e(\boldsymbol{\pi})\ell_e(y_e(\boldsymbol{\pi}))$, où $y_e(\boldsymbol{\pi}) = \sum_{k\in\mathcal{K}} \delta_{\pi_k}^e \lambda_k$ est le trafic total transitant sur le lien e avec la stratégie $\boldsymbol{\pi}$. Le problème d'optimisation peut alors s'écrire ainsi :

$$\text{minimiser } F(\boldsymbol{\pi}) = \sum_{e \in E} y_e(\boldsymbol{\pi})\, \ell_e(y_e(\boldsymbol{\pi})) \tag{OPT-R}$$

sous la contrainte :

$$\pi_k \in \Pi_k \qquad k \in \mathcal{K}. \tag{5.1}$$

Notons que ce problème peut s'écrire sous la forme d'un problème binaire en introduisant les variables de décision suivantes :

$$x_{k,\pi} = \begin{cases} 1 & \text{si la demande } k \text{ est routée sur le chemin } \pi \\ 0 & \text{sinon.} \end{cases} \tag{5.2}$$

En effet, on peut alors réécrire le problème (OPT-R) comme suit :

$$\text{minimiser } \sum_{e \in E} y_e \ell_e(y_e)$$

sous la contrainte :

$$y_e = \sum_{k \in \mathcal{K}} \sum_{\pi \in \Pi_k} \lambda_k \delta_\pi^e x_{k,\pi} \qquad e \in E, \tag{5.3}$$

$$\sum_{\pi \in \Pi_k} x_{k,\pi} = 1 \qquad k \in \mathcal{K}, \tag{5.4}$$

$$x_{k,\pi} \in \{0,1\} \qquad \pi \in \Pi_k, k \in \mathcal{K}. \tag{5.5}$$

5.3 Algorithme Best-response

Dans cette section, nous présentons l'algorithme proposé et certaines de ses propriétés. L'algorithme de meilleure réponse (Best Response) que nous proposons s'inspire d'un algorithme du même nom en théorie des jeux [51]. Dans un jeu, la meilleure réponse d'un joueur est définie comme sa stratégie optimale étant donnée la stratégie des autres joueurs. L'algorithme best-response considère plusieurs joueurs jouant à tour de rôle dans un ordre donné, chacun adaptant sa stratégie à son tour en fonction de la stratégie la plus récente des autres joueurs, jusqu'à ce qu'un point d'équilibre soit atteint.

Nous concevons le jeu de manière à ce que son équilibre de Nash fournisse une approximation efficace du problème (OPT-R). Nous suivons pour cela l'approche proposée dans [34]. Chaque joueur correspond à une demande en trafic. La stratégie du joueur k est le chemin π_k qu'il choisit dans l'ensemble Π_k, et un profil de stratégies du jeu est un vecteur $\boldsymbol{\pi} \in \Pi_1 \times \Pi_2 \times \ldots \times \Pi_K$. Un profil de stratégies correspond ainsi à une solution admissible du problème (OPT-R). Étant donnée la stratégie des autres joueurs

$\boldsymbol{\pi}_{-i} = (\pi_1, \pi_2, \ldots, \pi_{i-1}, \pi_{i+1}, \ldots, \pi_K)$, la valeur $f_i(\pi, \boldsymbol{\pi}_{-i})$ associée au chemin $\pi \in \Pi_i$ par le joueur i reflète le coût de ce chemin,

$$f_i(\pi, \boldsymbol{\pi}_{-i}) = \sum_{e \in \pi} \lambda_i \ell_e(y_e(\pi, \boldsymbol{\pi}_{-i})). \tag{5.6}$$

On note que la valeur associée à un chemin par le joueur i dépend de la stratégie des autres joueurs. On remarque également que

$$F(\boldsymbol{\pi}) = \sum_e \left(\sum_k \delta^e_{\pi_k} \lambda_k \right) y_e(\boldsymbol{\pi}) = \sum_k \lambda_k \sum_{e \in \pi_k} y_e(\boldsymbol{\pi}) = \sum_i f_i(\boldsymbol{\pi}).$$

Suivant l'approche proposée dans [34], afin de guider les joueurs vers un minimum local de $F(\boldsymbol{\pi})$, nous ajoutons un terme de pénalité $p_i(\boldsymbol{\pi})$ au coût du joueur i. Cette pénalité incite chaque joueur à prendre en compte l'impact de son action sur les autres joueurs. Quand le joueur i route son trafic sur le chemin π_i, l'augmentation du coût de chaque joueur $j \neq i$ utilisant les liens $e \in \pi_i$ est

$$\lambda_j \left[\ell_e \left(y_e^{-i} + \lambda_i \right) - \ell_e \left(y_e^{-i} \right) \right], \tag{5.7}$$

où $y_e^{-i} = \sum_{k \neq i} \delta^e_{\pi_k} \lambda_k$ représente le trafic total transitant sur le lien e en raison des joueurs différents de i. Ainsi, nous définissons le terme de pénalité comme suit :

$$p_i(\boldsymbol{\pi}) = \sum_{j \neq i} \lambda_j \sum_{e \in \pi_i} \delta^e_{\pi_j} \left[\ell_e \left(y_e^{-i} + \lambda_i \right) - \ell_e \left(y_e^{-i} \right) \right]. \tag{5.8}$$

En résumé, le joueur i calcule son chemin pour minimiser son propre coût, ce qui signifie qu'il résout le problème suivant :

$$\text{minimiser}_{\pi \in \Pi_i} c_i(\pi, \boldsymbol{\pi}_{-i}) = f_i(\pi, \boldsymbol{\pi}_{-i}) + p_i(\pi, \boldsymbol{\pi}_{-i}). \tag{OPT-i}$$

Dans (OPT-i), le chemin π minimisant $c_i(\pi, \boldsymbol{\pi}_{-i})$ est appelé la meilleure réponse (best-response) du joueur i étant donné la stratégie $\boldsymbol{\pi}_{-i}$ des autres joueurs. Le jeu est dans un équilibre de Nash si et seulement si la stratégie courante de chaque joueur est sa meilleure réponse à la stratégie des autres, impliquant qu'aucun joueur n'a intérêt à dévier unilatéralement de sa stratégie courante. Formellement, $\boldsymbol{\pi}$ est un équilibre de Nash si et seulement si

$$c_i(\boldsymbol{\pi}) \leq c_i(\pi, \boldsymbol{\pi}_{-i}), \forall \pi \in \Pi_i, \forall i \in \mathcal{K}.$$

Comme nous le montrerons, l'algorithme best-response que nous proposons comme approximation du problème (OPT-R) converge vers un équilibre de Nash. L'algorithme démarre de n'importequelle solution admissible $\boldsymbol{\pi}^{(0)}$. À chaque itération, tous les joueurs mettent à jour leur stratégie dans un ordre donné. Pour cela, chaque joueur calcule à

son tour la solution optimale du problème (OPT-i) en utilisant un algorithme de plus court chemin (l'algorithme de Dijkstra par exemple). On note qu'un joueur i dévie de sa stratégie $\pi_i^{(n)}$ à l'itération n pour une nouvelle stratégie π' si et seulement si $c_i\left(\pi', \boldsymbol{\pi}_{-i}^{(n)}\right) < c_i\left(\boldsymbol{\pi}^{(n)}\right)$. L'algorithme s'arrête lorsque $\boldsymbol{\pi}^{(n+1)} = \boldsymbol{\pi}^{(n)}$, autrement dit lorsque plus aucun joueur ne peut réduire son coût de façon unilatérale en modifiant sa stratégie. Le pseudo-code de l'heuristique est présenté dans l'Algorithme 3.

Algorithme 3 Best-response pénalisé.

Require: $\boldsymbol{\pi}^{(0)}$
1: $n \leftarrow 0$
2: **repeat**
3: **for** $i = 1 \dots K$ **do**
4: $\pi_i^{(n+1)} \leftarrow$ argmin (OPT-i)
5: **end for**
6: $n \leftarrow n + 1$
7: **until** $\boldsymbol{\pi}^{(n+1)} \neq \boldsymbol{\pi}^{(n)}$.
8: **return** $\vec{\pi}(n)$

Dans la suite, nous nous référerons à cet algorithme en parlant d'algorithme best-response pénalisé. L'algorithme de best-response non pénalisé correspond en fait au cas où chaque joueur i optimise directement la fonction $f_i(\pi, \boldsymbol{\pi}_{-i})$ et non la fonction pénalisée $c_i(\pi, \boldsymbol{\pi}_{-i}) = f_i(\pi, \boldsymbol{\pi}_{-i}) + p_i(\pi, \boldsymbol{\pi}_{-i})$. Comme nous le montrons ci-dessous, l'algorithme best-response pénalisé converge en un nombre fini d'itérations, alors qu'il n'y a pas de preuve de convergence connue pour la version non pénalisée.

5.3.1 Convergence de l'algorithme best-response pénalisé

Comme prouvé dans [34], la convergence de l'algorithme best-response pénalisé provient directement du fait que F est une fonction de potentiel du jeu. Nous décrivons brièvement la preuve dans ce qui suit. L'argument principal repose sur le lemme suivant.

Lemme 1. *La fonction objectif du joueur i peut s'écrire*

$$c_i(\boldsymbol{\pi}) = F(\boldsymbol{\pi}) - h_i(\boldsymbol{\pi}_{-i}), \tag{5.9}$$

où

$$h_i(\boldsymbol{\pi}_{-i}) = \sum_{e \in E} y_e^{-i}\, \ell_e(y_e^{-i}) \tag{5.10}$$

Démonstration. On note que $y_e(\boldsymbol{\pi}) = y_e^{-i}$ pour $e \notin \pi_i$, et $y_e(\boldsymbol{\pi}) = y_e^{-i} + \lambda_i$ pour $e \in \pi_i$.

D'où

$$\begin{aligned}
c_i(\boldsymbol{\pi}) &= f_i(\pi, \boldsymbol{\pi}_{-i}) + p_i(\pi, \boldsymbol{\pi}_{-i}), \\
&= \sum_{e \in \pi_i} \lambda_i \ell_e(y_e^{-i} + \lambda_i) + \sum_{e \in \pi_i} \sum_{j \neq i} \lambda_j \delta_{\pi_j}^e \left[\ell_e\left(y_e^{-i} + \lambda_i\right) - \ell_e\left(y_e^{-i}\right) \right], \\
&= \sum_{e \in \pi_i} (y_e^{-i} + \lambda_i)\, \ell_e(y_e^{-i} + \lambda_i) - \sum_{e \in \pi_i} y_e^{-i} \ell_e\left(y_e^{-i}\right), \\
&= \left(\sum_{e \in \pi_i} y_e(\boldsymbol{\pi})\, \ell_e(y_e(\boldsymbol{\pi})) + \sum_{e \notin \pi_i} y_e(\boldsymbol{\pi})\, \ell_e(y_e(\boldsymbol{\pi})) \right) - \left(\sum_{e \in \pi_i} y_e^{-i} \ell_e\left(y_e^{-i}\right) + \sum_{e \notin \pi_i} y_e^{-i} \ell_e\left(y_e^{-i}\right) \right), \\
&= F(\boldsymbol{\pi}) - h_i(\boldsymbol{\pi}_{-i}).
\end{aligned}$$

□

Proposition 2. *L'algorithme best-response pénalisé converge en un nombre fini d'itérations.*

Démonstration. Une conséquence directe du Lemme 1 est que la fonction F est une fonction de potentiel du jeu, autrement dit,

$$c_i(\boldsymbol{\pi}) - c_i(\pi, \boldsymbol{\pi}_{-i}) = F(\boldsymbol{\pi}) - F(\pi, \boldsymbol{\pi}_{-i}), \forall \pi \in \Pi_i, \forall i \in \mathcal{K},$$

impliquant que tout mouvement constituant une meilleure réponse pour un joueur résulte en une réduction du coût global $F(\boldsymbol{\pi})$. Comme $F(\boldsymbol{\pi})$ peut prendre un nombre fini de valeurs, la séquence atteindra un minimum local en un nombre fini d'itérations. □

Une autre conséquence du Lemme 1 est que tout optimum global est un équilibre de Nash, et donc que l'algorithme peut converger vers un optimum global en fonction du point de départ.

Proposition 3. *Tout optimum global est un équilibre de Nash du jeu de routage.*

Démonstration. Considérons un optimum global $\boldsymbol{\pi}^*$, i.e., un point tel que $F(\boldsymbol{\pi}^*) \leq F(\boldsymbol{\pi})$ pour tout $\boldsymbol{\pi} \in \times_i \Pi_i$. On suppose au contraire que $\boldsymbol{\pi}^*$ n'est pas un équilibre de Nash. Il s'ensuit directement qu'il existe un joueur i et une stratégie $\pi_i \in \Pi_i$ tels que $c_i(\pi_i, \boldsymbol{\pi}_{-i}^*) < c_i(\boldsymbol{\pi}^*)$. Avec le Lemme 1, cela implique que $F(\pi_i, \boldsymbol{\pi}_{-i}^*) - h_i(\boldsymbol{\pi}_{-i}^*) < F(\boldsymbol{\pi}^*) - h_i(\boldsymbol{\pi}_{-i}^*)$, à partir de quoi nous concluons que $F(\pi_i, \boldsymbol{\pi}_{-i}^*) < F(\boldsymbol{\pi}^*)$. Cela est clairement en contradiction avec l'optimalité globale de $\boldsymbol{\pi}^*$. Donc $\boldsymbol{\pi}^*$ est un équilibre de Nash. □

5.3.2 Garanties de performances

Un algorithme d'approximation retourne une solution à un problème d'optimisation combinatoire avec une erreur maximale garantie par rapport à l'optimum du problème

(contrairement à une heuristique qui ne fournit aucune garantie). Les algorithmes d'approximation sont typiquement utilisés quand la découverte de la solution optimale est impossible à atteindre. Le facteur d'approximation d'un algorithme est défini comme le supremum sur l'ensemble des instances du ratio entre le résultat obtenu par l'algorithme sur une instance et le coût optimal pour cette instance. Notre objectif dans cette section, est d'établir des bornes sur le facteur d'approximation de l'algorithme best-response pénalisé.

5.3.2.1 Fonction coût linéaire

Bien que, comme expliqué en introduction, notre motivation principale soit la résolution de problèmes non linéaires de routage mono-chemin, il est intéressant de considérer les performances de l'heuristique proposée pour une fonction linéaire de la forme $F(\boldsymbol{\pi}) = \sum_e a_e y_e(\boldsymbol{\pi})$, c'est-à-dire lorsque la fonction de latence $\ell_e(y)$ de chaque lien e est une constante positive a_e. Dans ce cas, comme expliqué ci-dessous, le problème de routage mono-chemin devient particulièrement simple puisqu'il devient séparable. La proposition 4 prouve que dans ce cas, l'heuristique proposée fournit nécessairement une solution optimale.

Proposition 4. *Si $F(\boldsymbol{\pi}) = \sum_{e\in E} a_e y_e$, l'algorithme de best-response pénalisé est optimal.*

Démonstration. On a

$$F(\boldsymbol{\pi}) = \sum_{e\in E} a_e y_e(\boldsymbol{\pi}) = \sum_{e\in E} a_e \sum_{k\in\mathcal{K}} \delta^e_{\pi_k}\lambda_k = \sum_{k\in\mathcal{K}} \lambda_k \left(\sum_{e\in\pi_k} a_e\right),$$

et la solution optimale est évidemment de router chaque flot k sur le chemin $\pi_k \in \Pi_k$ minimisant $\sum_{e\in\pi_k} a_e$, qui peut être vu comme la longueur du chemin. Il suffit donc de vérifier que l'heuristique proposée fait la même chose. Avec (5.8), on vérifie aisément que $p_i(\boldsymbol{\pi}) = 0$, d'où l'on déduit que $c_i(\pi, \boldsymbol{\pi}_{-i}) = f_i(\pi, \boldsymbol{\pi}_{-i}) = \lambda_i \left(\sum_{e\in\pi_i} a_e\right)$. On conclut ainsi que, comme la solution optimale, l'heuristique proposée route chaque demande sur un chemin de longueur minimale au sens des poids a_e. □

5.3.2.2 Fonction coût quadratique

Nous considérons dans ce paragraphe le cas où la fonction de latence $\ell_e(y)$ de chaque lien e est une fonction linéaire de la forme $\ell_e(y) = a_e y + b_e$, conduisant ainsi à un coût global quadratique $F(\boldsymbol{\pi}) = \sum_{e\in E} a_e y_e^2 + b_e y_e$. La Proposition 5 ci-dessous établit une borne supérieure sur le ratio d'approximation obtenu avec notre algorithme.

Proposition 5. *Si $F(\boldsymbol{\pi}) = \sum_{e\in E} y_e(a_e y_e + b_e)$, alors le facteur d'approximation de l'algorithme du best-response pénalisé est borné en dessus par $4 + 2\sqrt{3}$.*

Démonstration. Soit $\boldsymbol{\pi}$ une solution calculée par l'algorithme du best-response pénalisé et $\boldsymbol{\pi}^*$ une solution optimale. Dans ce qui suit, pour simplifier les notations, on note y_e et y_e^* les volumes de trafic $y_e(\boldsymbol{\pi})$ et $y_e(\boldsymbol{\pi}^*)$, respectivement. Comme $\boldsymbol{\pi}$ est un équilibre de Nash, on a $c_i(\boldsymbol{\pi}) \leq c_i(\pi_i^*, \boldsymbol{\pi}_{-i})$, pour tout $i \in \mathcal{K}$, ce qui d'après le Lemme 1 se traduit par $F(\boldsymbol{\pi}) \leq F(\pi_i^*, \boldsymbol{\pi}_{-i})$ pour tout i. La déviation du joueur i du chemin π_i vers le chemin π_i^* partitionne l'ensemble des liens en trois sous-ensembles séparés : ceux pour lesquels la charge augmente ($e \in \pi_i^* \setminus \pi_i$), ceux pour lesquels elle décroît ($e \in \pi_i \setminus \pi_i^*$), et enfin ceux pour lesquels elle reste constante. Nous définissons par conséquent

$$\alpha_e^i = \begin{cases} 1 & \text{si } e \in \pi_i^* \setminus \pi_i, \\ 0 & \text{sinon.} \end{cases} \quad \text{et} \quad \beta_e^i = \begin{cases} 1 & \text{si } e \in \pi_i \setminus \pi_i^*, \\ 0 & \text{sinon.} \end{cases}$$

En exploitant la linéarité de la fonction ℓ_e, la condition $F(\boldsymbol{\pi}) \leq F(\pi_i^*, \boldsymbol{\pi}_{-i})$ peut s'écrire comme suit :

$$\begin{aligned} F(\boldsymbol{\pi}) &\leq \sum_e \left(y_e + (\alpha_e^i - \beta_e^i)\lambda_i\right) \ell_e\left(y_e + (\alpha_e^i - \beta_e^i)\lambda_i\right) \\ &\leq F(\boldsymbol{\pi}) + \sum_e (\alpha_e^i - \beta_e^i)\lambda_i\, \ell_e(y_e) + y_e\, \ell_e\left((\alpha_e^i - \beta_e^i)\lambda_i\right) + (\alpha_e^i - \beta_e^i)\lambda_i\, \ell_e\left((\alpha_e^i - \beta_e^i)\lambda_i\right), \end{aligned}$$

de sorte que

$$-\sum_e (\alpha_e^i - \beta_e^i)\lambda_i\, \ell_e(y_e) \leq \sum_e y_e\, \ell_e\left((\alpha_e^i - \beta_e^i)\lambda_i\right) + (\alpha_e^i - \beta_e^i)\lambda_i\, \ell_e\left((\alpha_e^i - \beta_e^i)\lambda_i\right), \quad (5.11)$$

pour tout $i \in \mathcal{K}$. Comme

$$\begin{aligned} \sum_{i\in\mathcal{K}} (\alpha_e^i - \beta_e^i)\lambda_i &= \sum_{i\in\mathcal{K}} \left(\delta_{\pi_i^*}^e(1 - \delta_{\pi_i}^e) - \delta_{\pi_i}^e(1 - \delta_{\pi_i^*}^e)\right)\lambda_i, & (5.12) \\ &= \sum_{i\in\mathcal{K}} \delta_{\pi_i^*}^e \lambda_i - \sum_{i\in\mathcal{K}} \delta_{\pi_i}^e \lambda_i, & (5.13) \\ &= y_e^* - y_e, & (5.14) \end{aligned}$$

en sommant sur i les inégalités (5.11) on trouve

$$F(\boldsymbol{\pi}) \leq \sum_e y_e^* \ell_e(y_e) + y_e\, (a_e(y_e^* - y_e) + b_e) + a_e \sum_i \left((\alpha_e^i - \beta_e^i)\lambda_i\right)^2 + b_e(y_e^* - y_e), \quad (5.15)$$

On note que comme $\alpha_e^i\, \beta_e^i = 0$, nous avons

$$\begin{aligned}\sum_e a_e \sum_i \left((\alpha_e^i - \beta_e^i)\lambda_i\right)^2 &= \sum_e a_e \sum_i (\alpha_e^i \lambda_i)^2 + \sum_e a_e \sum_i (\beta_e^i \lambda_i)^2 - 2\sum_e a_e \sum_i \alpha_e^i \beta_e^i \lambda_i^2, \\ &\leq \sum_e a_e \left(\sum_i \alpha_e^i \lambda_i\right)^2 + \sum_e a_e \left(\sum_i \beta_e^i \lambda_i\right)^2, \\ &\leq \sum_e a_e (y_e^*)^2 + \sum_e a_e (y_e)^2,\end{aligned}$$

où la dernière égalité provient directement de $\sum_i \alpha_e^i \lambda_i \leq \sum_i \delta_{\pi_i^*}^e \lambda_i = y_e^*$ (et de manière similaire pour y_e). En conséquence, (5.15) peut s'écrire

$$\begin{aligned}F(\boldsymbol{\pi}) &\leq \sum_e y_e^*(a_e y_e + b_e) + a_e y_e y_e^* + b_e y_e - a_e (y_e)^2 + a_e (y_e^*)^2 + + a_e (y_e)^2 + b_e y_e^* - b_e y_e, \\ &\leq F(\boldsymbol{\pi}^*) + 2\sum_e a_e y_e y_e^* + \sum_e b_e y_e^*. \qquad (5.16)\end{aligned}$$

On note que $\sum_e b_e y_e^* \leq \sum_e y_e^*(a_e y_e^* + b_e) \leq F(\boldsymbol{\pi}^*)$ et une application directe de l'inégalité de Cauchy-Schwartz amène à

$$\left(\sum_e a_e y_e y_e^*\right)^2 \leq \sum_e a_e (y_e)^2 \sum_e a_e (y_e^*)^2 \leq F(\boldsymbol{\pi}) F(\boldsymbol{\pi}^*).$$

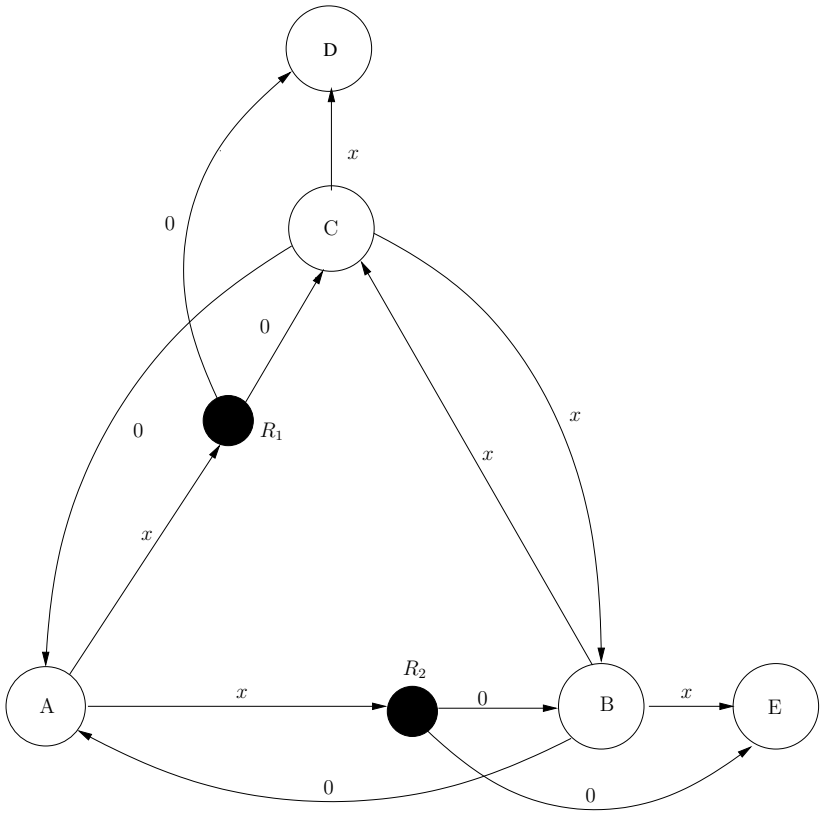
En définissant $x = \sqrt{\frac{F(\boldsymbol{\pi})}{F(\boldsymbol{\pi}^*)}}$, l'équation (5.16) amène à $x^2 \leq 2x + 2$, à partir de quoi nous concluons que $x \leq 1 + \sqrt{3}$, et donc que $F(\boldsymbol{\pi}) \leq \left[4 + 2\sqrt{3}\right] F(\boldsymbol{\pi}^*)$. □

Remarque 5.3.1. *Si $b_e = 0$ pour tout e, i.e., si $F(\boldsymbol{\pi}) = \sum_{e \in E} a_e (y_e)^2$, alors (5.16) devient $F(\boldsymbol{\pi}) \leq F(\boldsymbol{\pi}^*) + 2\sqrt{F(\boldsymbol{\pi}) F(\boldsymbol{\pi}^*)}$, d'où l'on déduit que $F(\boldsymbol{\pi}) \leq \left[3 + 2\sqrt{2}\right] F(\boldsymbol{\pi}^*)$.*

On note que la borne supérieure obtenue pour l'algorithme de best-response pénalisé est nettement supérieure à celle obtenue dans [18] pour l'algorithme de best-response non pénalisé, qui est seulement de $\frac{3+\sqrt{5}}{2} \approx 2,618$. Les auteurs de [18] prouvent de plus que leur borne est serrée, dans le sens où ils fournissent une instance du problème pour laquelle elle est atteinte. Comme le montre la Proposition 5, en s'inspirant de l'exemple utilisé dans [18] pour l'obtention d'une borne inférieure, on peut construire une instance du problème pour laquelle le ratio d'approximation du best-response pénalisé est strictement supérieur à $\frac{3+\sqrt{5}}{2}$.

Proposition 6. *Si $F(\boldsymbol{\pi}) = \sum_{e \in E} y_e(a_e y_e + b_e)$, alors le facteur d'approximation de l'algorithme du best-response pénalisé est borné par en dessous par $2 + \sqrt{2} \approx 3,414$.*

Démonstration. Nous considérons l'instance représentée sur la Figure 5.2. Le réseau comprend 7 nœuds, 12 liens et 4 flots OD. La figure précise l'origine, la destination et la demande de chaque flot. Elle indique également le routage de chaque flot à l'optimum (OPT) et à l'équilibre de Nash (NE). Nous donnons dans la Table 5.1 le trafic sur chaque lien et le coût associé pour chacune des deux solutions (pour simplifier, seuls les liens tels que $\ell_e(x) \neq 0$ sont présentés).



	NE	OPT
Flow 1 : $A \to D$, $\lambda_1 = \phi$	$A \to R_2 \to B \to C \to D$	$A \to R_1 \to D$
Flow 2 : $A \to E$, $\lambda_2 = \phi$	$A \to R_1 \to C \to B \to E$	$A \to R_2 \to E$
Flow 3 : $C \to B$, $\lambda_3 = 1$	$C \to A \to R_2 \to B$	$C \to B$
Flow 4 : $B \to C$, $\lambda_4 = 1$	$B \to A \to R_1 \to C$	$B \to C$

FIGURE 5.2 – Instance considérée pour la borne inférieure sur le ratio d'approximation.

On vérifie aisément l'optimalité de la stratégie de routage décrite sur la figure. En effet, dans cette stratégie, chaque flot est routé sur un lien de coût $\ell_e(x) = x$, sur lequel il est tout seul, et éventuellement sur un autre lien de coût $\ell_e(x) = 0$. Il n'est clairement pas possible de faire mieux.

Pour vérifier que le routage proposé en tant qu'équilibre de Nash correspond bien à

	OPT			NE		
Lien	Flots	Trafic	Coût	Flots	Trafic	Coût
A,R1	1	ϕ	ϕ^2	2,4	$\phi+1$	$(\phi+1)^2$
C,D		0	0	1	ϕ	ϕ^2
A,R2	2	ϕ	ϕ^2	1,3	$\phi+1$	$(\phi+1)^2$
C,B	3	1	1	2	ϕ	ϕ^2
B,C	4	1	1	1	ϕ	ϕ^2
B,E		0	0	2	ϕ	ϕ^2

TABLE 5.1 – Trafic sur les liens et coût associé à l'optimum et à l'équilibre de Nash.

un équilibre, il suffit, d'après le Lemme 5.9, de vérifier qu'on ne peut pas faire décroître le coût global du réseau en reroutant un seul flot. Nous illustrons la démarche en prenant le cas du flot 1. Si nous déroutons ce flot du chemin $A \to R2 \to B \to C \to D$ vers le chemin $A \to R1 \to D$, le trafic sur les liens $(A, R1)$ et $(R1, D)$ augmente de ϕ, et celui sur les liens $(A, R2)$, $(R2, B)$, (B, C) et (C, D) diminue d'autant. Le coût des autres liens ne change pas. Avec la Table 5.1, on voit de suite que la variation de coût est

$$\left[(2\phi+1)^2+0+1+0+0+0\right]-\left[(\phi+1)^2+0+(\phi+1)^2+0+\phi^2+\phi^2\right]=0.$$

On vérifiera aisément que, de la même manière, le coût global ne peut diminuer en déplaçant un des autres flots. On peut d'ailleurs noter que l'exemple est construit de telle manière que l'équilibre est un équilibre faible, c'est-à-dire que pour tout joueur i, il existe un chemin π différent du chemin π_i utilisé à l'équilibre et tel que $c_i(\pi, \boldsymbol{\pi}_{-i}) = c_i(\boldsymbol{\pi})$.

En sommant les coûts des liens dans la Table 5.1, on peut vérifier que le coût à l'optimum est $F(\boldsymbol{\pi}^*) = 2\phi^2 + 2$, alors que le coût à l'équilibre de Nash est $F(\boldsymbol{\pi}) = 2(\phi+1)^2 + 4\phi^2 = 6\phi^2 + 4\phi + 2$. Le ratio d'approximation est donc

$$g(\phi) = \frac{6\phi^2+4\phi+2}{2\phi^2+2}.$$

En dérivant, on vérifie aisément que le maximum est atteint pour $\phi = 1+\sqrt{2}$, et que ce maximum correspond à $2+\sqrt{2}$.

□

On peut faire plusieurs remarques concernant le résultat ci-dessus. Tout d'abord, on note que la borne inférieure de $3,414$ obtenue est assez éloignée de la borne supérieure de $3+2\sqrt{2} = 5,83$ de la remarque 5.3.1. Intuitivement, au vu de certaines inégalités utilisées dans la démonstration de la Proposition 5, on peut penser que c'est cette borne supérieure qui est trop large. On remarque aussi que la Proposition 6 prouve que le ratio d'approximation pire-cas du best-response pénalisé est nettement supérieur à la valeur

2,618 de celui du best-response non pénalisé. Comme nous l'évoquerons au paragraphe 5.5, les deux algorithmes ont en pratique des performances très proches, l'algorithme pénalisé ayant l'avantage d'avoir une convergence garantie.

5.4 Quelques algorithmes existants

Dans cette partie, nous présentons deux techniques existantes qui peuvent être utilisées pour résoudre (OPT-R). Nous utiliserons ces algorithmes à des fins de comparaison.

5.4.1 Global Smoothing Algorithm (GSA)

Cet algorithme récent, introduit dans [82], est conçu pour prendre en charge des problèmes d'optimisation non-linéaire avec des variables binaires. D'après les résultats exposés par ces auteurs, il semble capable de fournir de bonnes solutions entières dans des temps raisonnables. Nous comparerons les résultats fournis par cet algorithme avec ceux de l'algorithme de meilleure réponse.

Son approche consiste à considérer une séquence d'optimisation de problèmes non-linéaires en variables continues. Cette méthode optimise une fonction coût pénalisée $F(\mathbf{x})$, obtenue en ajoutant deux fonctions de pénalité à la fonction coût $\phi(\mathbf{x})$ du problème originel : une barrière logarithmique $L(\mathbf{x})$ et une pénalité "exacte" $G(\mathbf{x})$:

$$F(\mathbf{x}) = \phi(\mathbf{x}) + L(\mathbf{x}) + G(\mathbf{x}) \tag{5.17}$$

$L(\mathbf{x})$ est conçue pour lisser la fonction coût originelle et pour garder le point courant suffisamment loin de la grille des nombres entiers : si une valeur x_i devient trop proche de 0 ou de 1, la fonction tend vers l'infini. Appliqué à notre problème (OPT-R), la formulation de cette pénalité devient :

$$L(\mathbf{x}) = -\mu \sum_{k \in K} \sum_{\pi \in \Pi_k} \left(\ln(\frac{x_k^\pi}{\lambda_k}) + \ln(1 - \frac{x_k^\pi}{\lambda_k}) \right) \tag{5.18}$$

$L(\mathbf{x})$ est une fonction strictement convexe. Elle est associée avec un paramètre μ, choisi suffisamment grand en début d'optimisation puis diminué progressivement à chaque itération (le choix de ce paramètre sera décrit plus en détails dans la section 5.4.1.1. Il est important de noter que cette fonction n'est pas définie si la solution x comporte un flux routé sur un unique chemin ($x_k^\pi = 0$ ou $x_k^\pi = 1$), ce qui signifie que le point de départ choisi doit nécessairement être tel que tous les flux sont routés sur plusieurs chemins.

La fonction de pénalité "exacte" $G(\mathbf{x})$ est associée au paramètre γ, qui est choisi suffisamment petit en début d'optimisation, et augmente au cours des itérations successives,

afin de se rapprocher graduellement d'une solution entière :

$$G(\mathbf{x}) = \gamma \sum_{k \in K} \sum_{\pi \in \Pi_k} \left(\frac{x_k^\pi}{\lambda_k} (1 - \frac{x_k^\pi}{\lambda_k}) \right) \tag{5.19}$$

Les principales étapes de l'algorithme sont exposées dans l'Algorithme 4. L'idée derrière cette méthode est que la séquence de problèmes pénalisés puisse former une trajectoire amenant à une bonne approximation de la solution globale du problème originel avec variable binaire. Comme l'algorithme se résume à une séquence de minimisation de problèmes, il est donc très dépendant de l'algorithme utilisé pour chaque minimisation.

Nous avons décidé d'utiliser l'algorithme de gradient conjugué présenté dans [22], car il fournit une méthode efficace, supportant relativement bien le passage à l'échelle. Le pas maximal fourni par l'algorithme de recherche linéaire utilisé par le gradient conjugué est calculé de façon à ce que toutes les capacités des liens $c_e, e \in E$ soient toujours respectées.

Algorithme 4 GSA(Global Smoothing Algorithm).

1: $\mu \leftarrow \mu_0$
2: $\gamma \leftarrow \gamma_0$
3: $\mathbf{x} \leftarrow \mathbf{x}_0$
4: $0 < \theta_\mu < 1$ et $\theta_\gamma \leftarrow 1/\theta_\mu$
5: **while not** $xStart$ suffisamment proche de la grille des entiers **do**
6: $\quad x \leftarrow minimizeFunction(F, \mathbf{x}, \mu, \gamma)$
7: $\quad \mu = \theta_\mu \mu$
8: $\quad \gamma = \theta_\gamma \gamma$
9: **end while**

La trajectoire suivie par la séquence de minimisation est directement impactée par le choix des paramètres de pondération μ et γ associés aux pénalités. Nous rappelons dans la section suivante comment doivent être déterminés ces paramètres pour obtenir un résultat correct.

5.4.1.1 Choix de μ_0 et γ_0

Le choix initial de μ et γ est une problématique difficile abordée dans [82]. Il s'agit clairement de trouver le bon équilibre entre les deux pénalités et la fonction coût d'origine. Les auteurs proposent d'utiliser une valeur initial pour μ égale à la valeur propre maximale de la Hessienne de ϕ, ce choix étant suffisant pour garantir que la fonction soit strictement convexe. La vraie difficulté réside dans le choix initial du paramètre γ : quand il augmente, la fonction pénalisée devient non convexe, introduisant des minima locaux. La non-convexité de la pénalité ne doit pas faire oublier la fonction coût dans les premières étapes de l'algorithme. Les auteurs de l'algorithme préconisent l'utilisation d'un valeur initiale de γ égale à 1% de la valeur absolue de la valeur propre minimale

de la Hessienne de la fonction $\phi(\mathbf{x}) + L(\mathbf{x})$.

Le taux de réduction de μ est habituellement choisi dans l'intervalle $[0.1, 0.9]$, tandis que le taux de décroissance de γ est choisi tel que $\theta_\mu \theta_\gamma = 1$.

5.4.1.2 Évolution de la forme de la fonction coût pénalisée

Pour mieux comprendre comment agissent les pénalités sur la fonction coût, nous proposons d'observer la fonction coût pénalisée sur un exemple très simple de routage. On considère une topologie comportant 2 nœuds A et B, et deux liens parallèles entre A et B. Les capacités des deux liens sont différentes : $c_1 = 6Mbps$ et $c_2 = 8Mbps$. On souhaite faire passer une demande de valeur d du nœud A au nœud B.

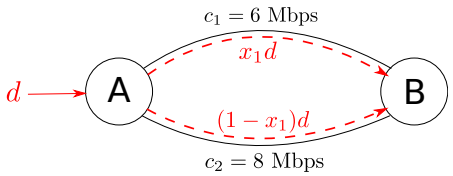


FIGURE 5.3 – Exemple simple pour illustrer la forme de la fonction pénalisée.

Si l'on considère qu'un flux peut-être routé sur plusieurs chemins en parallèle, la demande totale circulant sur le lien du haut vaut alors $y_1 = x_1 d$. Le reste de la demande est routée sur le lien du bas : $y_2 = (1 - x_1)d$. Il s'agit alors d'un problème comportant une seule variable x_1.

On considère la fonction coût quadratique $f(x) = \left(\frac{y_1}{c_1}\right)^2 + \left(\frac{y_2}{c_2}\right)^2$. Nous traçons sur la Figure 5.4 la forme de la fonction coût pénalisée pour un ensemble de valeurs μ et γ (sur la figure m représente μ, et g représente γ) représentatives de leur évolution durant l'exécution de l'algorithme. Sur chaque figure, la courbe jaune correspond à la fonction coût non pénalisée ($\gamma = 0$ et μ=0). Les flèches rouges entre les figures représentent le sens de parcours suivi par l'algorithme.

Au début de l'algorithme la fonction coût est très clairement convexe (dominance de la barrière logarithmique). À la fin de l'algorithme, la fonction est quasiment rendue concave, ce qui repousse les solutions vers les valeurs binaires.
Nous renvoyons le lecteur vers [82] pour de plus amples détails sur la méthode GSA.

5.4.2 Optimisation par colonie de fourmis

Cette méta-heuristique appartient à la famille des algorithmes de type optimisation par colonie de fourmis, que nous avons évoqué dans le chapitre 3. Toutes les fourmis laissent des traces de phéromones sur leur passage, et sont également capables de les détecter. Les fourmis ont tendance à suivre le chemin pour lequel la trace de phéromones

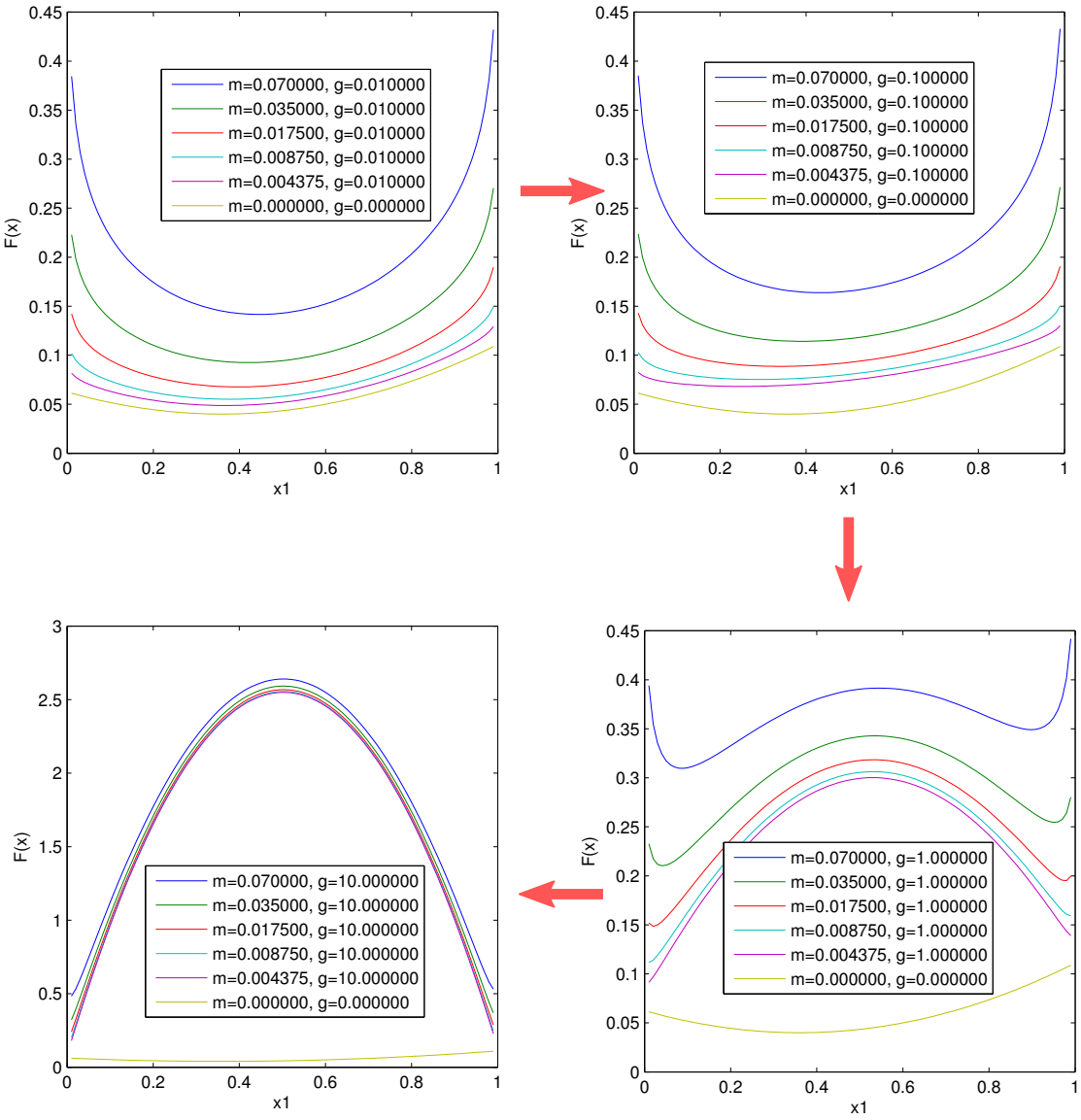


FIGURE 5.4 – Évolution de la forme de la fonction pénalisée manipulé par l'algorithme GSA.

est la plus importante. Elles utilisent donc ces traces comme moyen de partage d'infor-

mations, ce qui leur apporte une forme d'intelligence collective. Les traces s'évaporant rapidement, elles perdurent plus longtemps sur les chemins les plus utilisés.

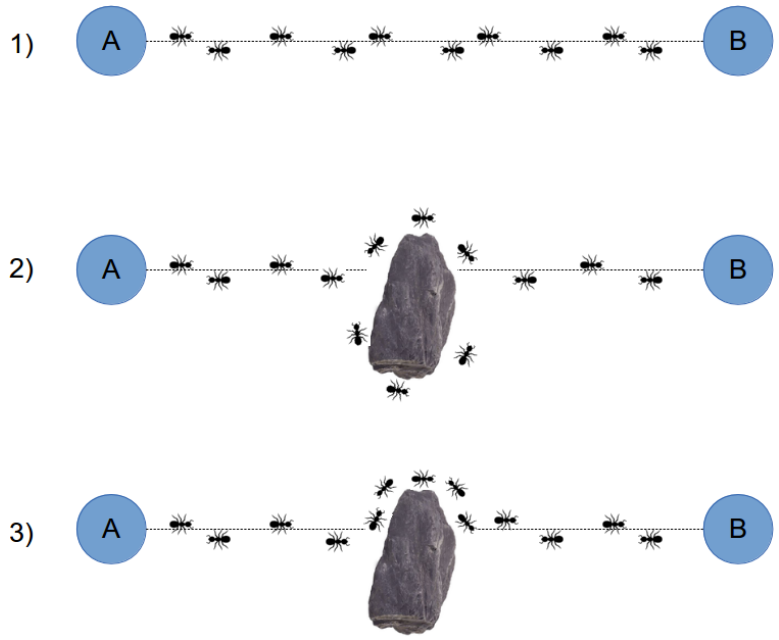


FIGURE 5.5 – Comportement collectif des fourmis.

Nous représentons sur la Figure 5.5 le comportement d'un groupe de fourmis entre deux points A et B. À l'étape 1), les fourmis parcourent le chemin direct A-B. Un obstacle apparaît sur ce chemin à l'étape 2. Les fourmis le contournent de manière aléatoire, tout en laissant des traces de phéromones sur leur passage. Le chemin étant plus court par le haut, les traces de phéromones vont progressivement devenir plus fortes en haut qu'en bas. Enfin, dans l'étape 3, l'intégralité des fourmis a convergé vers le plus court chemin, grâce à l'accumulation plus forte de phéromones sur le chemin le plus court.

L'algorithme que nous considérons ici est présenté dans [52]. Il a été conçu pour optimiser des réseaux MPLS et fournit de bons résultats d'approximation tout en conservant des temps d'exécution raisonnables. Trois grandes étapes se détachent de l'algorithme :

— déterminer les chemins possibles Π_k pour chaque flux k
— approximer une solution du problème multichemins (avec les chemins calculés à l'étape précédente) à l'aide d'un algorithme de type gradient projeté tel que celui présenté dans [99]
— lancer l'optimisation par colonie de fourmis, en utilisant le résultat de l'étape précédente pour initialiser les distributions de probabilités utilisés par les fourmis.

Dans la dernière étape, L fourmis sont définies, chacune d'entre elles explorant l'es-

pace des solutions en fonction de distributions de probabilité. À chaque itération, chaque fourmi $l \in L$ construit une solution complète en routant l'intégralité des $\mathcal{K}$ flux. Elle commence par choisir un LSP k de façon aléatoire (selon une loi uniforme), puis lui choisi une route π suivant une loi de probabilité dépendant des traces de phéromones sur les chemins envisageables Π_k. Elle route alors le flux k sur le chemin π, et met à jour la trace de phéromones en conséquence. Elle choisit alors un nouveau LSP à router et répète les étapes précédentes. La fourmi continue jusqu'à ce que les $\mathcal{K}$ flux soient routés. Les L fourmis procèdent ainsi à tour de rôle pour chaque itération.

L'algorithme complet s'arrête après un nombre maximum et fixé d'itérations. Le fonctionnement de l'heuristique permet d'obtenir une solution dès la première itération, ce qui est très intéressant si l'on souhaite trouver au moins une solution admissible, sans accorder d'importance à la qualité de cette dernière. Les itérations suivantes sont présentes pour raffiner la qualité de la première solution, si de meilleures solutions sont trouvées.

Dans la suite de ce document, nous nous référerons à cet algorithme par l'abréviation ACO(Ant Colony Optimization).

5.5 Résultats numériques d'optimisation du mono-routage

Nous analysons ici l'efficacité de l'algorithme de meilleure réponse (qui sera nommé BR dans la suite du document) sur deux fonctions coût non linéaires, et ce par rapport aux algorithmes GSA et ACO. La solution optimale du problème de routage multichemins est utilisée comme borne inférieure à partir de laquelle nous calculons une "erreur" relative.

Les simulations sont effectuées sur 8 topologies (voir Table 5.2), trouvées dans la littérature IEEE et dans [108]. Étant donné la taille des problèmes et leur complexité, nous nous restreignons à au plus deux plus courts chemins (en termes de nombre de sauts) pour chaque paire source/destination. Pour chaque réseau, nous considérons un ensemble de 100 matrices de trafic aléatoires (comportant un flux entre chaque source/destination). Ces matrices sont admissibles, dans le sens où nous nous assurons qu'il existe un routage mono-chemin tel qu'aucune capacité de lien ne soit dépassé. Le nombre de variables associé à chaque scénario est visible dans la Table 5.2.

Pour GSA, deux solutions multi-routées ont été utilisées comme point de départ : un point aléatoire, et un point multi-routé très proche de la solution mono-routée obtenue avec l'algorithme de meilleure réponse. Les résultats présentés dans la prochaine partie se composent des meilleures solutions obtenues à partir de ces deux points de départ. Pour ACO, nous considérons au plus 50 itérations et 5 fourmis : nos expérimentations sur les scénarios considérés nous ont révélés qu'augmenter ces paramètres n'améliorait

Topologie	N	M	Variables binaires
ABOVENET	19	68	664
ARPANET	24	100	1104
BHVAC	19	46	684
EON	19	74	684
METRO	11	84	220
NSF	8	20	112
PACBELL	15	42	420
VNSL	9	22	140

TABLE 5.2 – Topologies : nombre de nœuds (**N**), liens (**M**) et nombre de variables binaires correspondant au scénario associé.

pas sensiblement les résultats obtenus, alors qu'ils impactent négativement les temps d'exécution.

Tous nos résultats ont été obtenus via une implémentation des algorithmes en Matlab 2013b, sur un processeur Intel Core i5-2430M tournant à 2.4 GHz, avec un système d'exploitation Linux 64 bits disposant de 4Go de mémoire vive.

5.5.1 Fonction coût quadratique

Nous considérons la fonction quadratique suivante :

$$F(\boldsymbol{\pi}) = \sum_{e \in E} \left(\frac{y_e(\boldsymbol{\pi})}{c_e} \right)^2 \tag{5.20}$$

Les résultats obtenus en optimisant cette fonction sont résumés dans la Table 5.3. Tout d'abord, nous remarquons que chaque algorithme fournit de bons résultats d'approximation avec cette fonction coût, car l'erreur relative moyenne reste inférieure à 4,37% sur l'ensemble de nos scénarios. Nous observons aussi que l'algorithme GSA semble fournir les meilleurs résultats d'optimisation. Mais en regardant les temps d'exécution présentés dans la Table 5.5, nous observons que l'algorithme BR est clairement plus rapide que GSA et ACO, son pire temps d'exécution restant inférieur à 6.8 secondes, là où l'algorithme ACO atteint 50 secondes et GSA environ 300 secondes.

5.5.2 Fonction coût M/M/1

Dans cette partie, nous étudions les résultats obtenus avec une fonction coût souvent utilisée dans le domaine de l'optimisation réseau : la fonction délai de Kleinrock [65], correspondant à une file d'attente de type M/M/1. La modélisation M/M/1 suppose que les paquets arrivent suivant une loi exponentielle, et qu'ils ont une durée de service elle

Topologie	BR			GSA			ACO		
	min	max	moy	min	max	moy	min	max	moy
ABOVENET	1,28	5,18	3.86	$\simeq 0$	13,32	7,19	0,17	4,87	2,98
ARPANET	1.05	2.83	1,84	1,09	5,40	2,57	4,42	8,49	5,99
BHVAC	0.24	1,16	0,65	0,09	4,35	0,94	3,93	8,30	5,78
EON	$\simeq 0$	3,37	2,31	$\simeq 0$	3,63	0,89	2,24	6,63	4,53
METRO	2,23	20,16	8.59	1,64	8,90	4,47	2,63	13,32	7,41
NSF	$\simeq 0$	12,97	3,29	0,04	4,43	1,76	0,15	7,46	2,21
PACBELL	1.89	5,27	3,61	0,32	3,00	1,44	2,30	8,00	4,76
VNSL	0,03	7,27	1.91	$\simeq 0$	8,57	3,07	0,09	4,92	1,34
Global	$\simeq 0$	20,16	3,26	$\simeq 0$	13,32	2,79	0,09	13,32	4,37

TABLE 5.3 – Erreurs relatives (%) par rapport au multichemins, avec fonction quadratique.

aussi exponentielle. On considère ici la fonction suivante :

$$F(\boldsymbol{\pi}) = \sum_{e \in E} \frac{y_e(\boldsymbol{\pi})}{c_e - y_e(\boldsymbol{\pi})} \tag{5.21}$$

Ce modèle est intéressant car il permet de trouver un routage minimisant le délai moyen dans le réseau.

Les résultats obtenus sont exposés dans la Table 5.4 : ACO et BR obtiennent ici des résultats meilleurs que GSA en termes de minimisation du coût. L'algorithme GSA semble rencontrer des difficultés sur certaines instances, avec une erreur relative importante. L'algorithme BR fournit la meilleure optimisation moyenne, tandis qu'ACO est celui qui minimise le mieux l'erreur maximale. Mais comme le montrent les résultats de la Table 5.5, BR offre à nouveau le meilleur compromis entre efficacité d'optimisation et temps d'exécution : son pire temps d'exécution, toutes instances confondues, reste inférieur à 4.7 secondes, là où GSA atteint 400 secondes et ACO 50 secondes.

Remarque 5.5.1. *Les erreurs relatives obtenues avec l'algorithme best-response non pénalisé sont comparables à celles que nous présentons ici pour l'algorithme pénalisé. En pratique on obtient des erreurs parfois meilleures, parfois moins bonnes, mais l'écart entre les deux méthodes est toujours inférieur à quelques %. Les temps d'exécutions sont par contre meilleurs avec l'algorithme non pénalisé : au maximum 1.1 secondes au pire cas pour toutes les instances. Cela est normal, car l'algorithme pénalisé doit calculer la pénalité pour chaque joueur.*

Topologie	BR			GSA			ACO		
	min	max	moy	min	max	moy	min	max	moy
ABOVENET	$\simeq 0$	1,52	0,72	$\simeq 0$	87,54	5,95	1,94	6,23	3,59
ARPANET	$\simeq 0$	0.50	0,13	$\simeq 0$	77,01	3,44	2,58	7,64	4,82
BHVAC	$\simeq 0$	1,45	0,35	$\simeq 0$	64,58	8,57	3,30	8,54	5,17
EON	$\simeq 0$	1.87	0,82	$\simeq 0$	58,09	3,15	1,71	8,92	3,63
METRO	$\simeq 0$	24.45	2.38	$\simeq 0$	30,89	3,62	0,20	14,99	2,77
NSF	$\simeq 0$	21.06	2.77	0,01	60,14	3,75	0,01	8,77	1,47
PACBELL	$\simeq 0$	1.19	0.20	$\simeq 0$	45,13	5,30	1,44	7,14	4,00
VNSL	$\simeq 0$	4.40	0.64	$\simeq 0$	24,07	3,51	0,34	6,82	2,46
Global	$\simeq 0$	24,45	1.00	$\simeq 0$	87,54	4,53	0,01	14,99	3,26

TABLE 5.4 – Erreurs relatives (%) par rapport au multichemins, avec fonction M/M/1.

Scénario	Fonction quadratique			Fonction M/M/1		
	BR	GSA	ACO	BR	GSA	ACO
Min	0,04	1,02	4,73	0,03	0,75	4,98
Max	6.75	297,73	48,83	4.68	400,82	50,95
Moyenne	0,92	30,85	22,16	0,72	27,45	22,40

TABLE 5.5 – Temps d'exécution (en secondes) pour l'ensemble des scénarios testés.

5.6 Optimisation dynamique du placement des LSP avec l'algorithme Best Response

Les résultats obtenus pour le placement initial de l'ensemble des LSP sont très intéressants, et nous font penser que l'algorithme proposé doit pouvoir être utilisé à des fins de reconfiguration dynamique des LSP. La vitesse avec laquelle l'algorithme proposé fournit une solution le rend particulièrement attractif dans le cadre d'une utilisation en ligne, où le temps d'optimisation doit être limité le plus possible. Une telle reconfiguration dynamique permettrait d'adapter automatiquement les chemins des LSP aux variations de trafic observées sur le réseau.

Ici, contrairement au cas de l'optimisation des poids OSPF, l'incertitude de mesure sur les trafics est bien moindre, car les trafics peuvent être mesurés LSP par LSP (en observant leurs interfaces via le protocole SNMP). Nous considérons pour nos simulations que nous disposons toutes les 5 minutes du trafic moyen de chaque LSP sur les 5 dernières minutes.

Afin d'illustrer l'utilisation de l'algorithme de meilleure réponse dans un cadre dynamique, nous avons effectué des simulations de variations de trafic sur les 8 topologies

testées précédemment (voir Table 5.2). Pour chaque topologie, nous choisissons une matrice de trafic aléatoire pleine à l'instant $\tau = 0$, comportant donc un LSP par couple source/destination. Chaque minute, la matrice de trafic est mise à jour via l'ajout d'un bruit gaussien sur la demande de chaque LSP k :

$$d_k^t = d_k^{t-1\,minute} + \mathcal{N}(0, \sigma^2) \quad k = 1, \ldots, K. \tag{5.22}$$

La déviation standard de ce bruit est choisie telle que 99.7% des demandes varient d'au plus ±8.5% par minute (et cette borne est forcée pour les 0.3% restants). En conséquence, chaque demande peut varier d'au plus ±50% sur un intervalle de 5 minutes :

$$0.5 d_k^{\tau - 5min} \leq d_k^{\tau} \leq 1.5 d_k^{\tau - 5min} \tag{5.23}$$

Toutes les 5 minutes, nous exploitons les informations de trafic moyennes de la dernière fenêtre temporelle (de 5 minutes) pour optimiser le placement des LSP. Afin de limiter le nombre le nombre de LSP reroutés par l'algorithme de BestResponse, nous introduisons un gain seuil G_{Seuil} conditionnant le choix de chaque joueur : un joueur ne change sa propre stratégie que si cela lui permet de diminuer son propre coût d'au moins G_{Seuil} par rapport à sa précédente stratégie. Pour ces simulations, nous fixons arbitrairement G_{Seuil} à 5% de gain minimum.

Nous considérons des simulations de 250 minutes (environ 4 heures), comportant donc 50 déclenchements de l'algorithme d'optimisation dynamique (toutes les 5 minutes).

5.6.1 Coûts moyens sur la période de simulation

Les Tables 5.6 et 5.7 résument les coûts moyens obtenus sur l'ensemble des topologies, pour la durée de la simulation, pour les deux types de fonction coût étudiées. Trois configurations y sont représentées pour chaque topologie :

— *Statique* : configuration statique de tous les LSP, basée sur un routage OSPF avec des métriques unitaires sur chaque lien. Les LSP sont alors routés sur les plus courts chemins entre leur source et leur destination.

— *Optim hors ligne* : résultats obtenus avec un placement optimisé hors ligne avec l'algorithme de Best Response, à partir de la matrice de trafic moyenne de l'ensemble de la simulation. Cette configuration est purement hypothétique, car elle suppose que l'on puisse prévoir à l'avance une matrice de trafic moyenne sur la période.

— *Optim dynamique* : obtenus en exécutant l'algorithme Best Response toutes les 5 minutes, en exploitant les informations moyennes de trafic des 5 dernières minutes. À l'instant $\tau = 0$, le placement des LSP correspond à la configuration statique (plus courts chemins).

Topologie	Statique	Optim hors-ligne	Optim dynamique
ABOVENET	1,59	1,43	1,39
ARPANET	4,68	4,44	4,21
BHVAC	2,52	2,43	2,44
EON	2,94	2,87	2,82
METRO	2,23	2.16	2.04
PACBELL	2,64	2,42	2,38
NSF	0,95	0,93	0,91
VNSL	2,68	2,65	2,64

TABLE 5.6 – Coût moyen sur la période de simulation avec la fonction quadratique sur les différentes topologies.

Topologie	Statique	Optim hors-ligne	Optim dynamique
ABOVENET	7,24	6,83	6,81
ARPANET	20,73	19,52	18.85
BHVAC	13,16	12,90	12,89
EON	15,70	15.36	15.08
METRO	7.27	6.98	6.89
PACBELL	10,49	9,85	9,75
NSF	4,37	4,20	4,19
VNSL	5,14	4.90	4,92

TABLE 5.7 – Coût moyen sur la période de simulation avec la fonction M/M/1 sur les différentes topologies.

On remarque la proximité entre les résultats obtenus avec l'algorithme en ligne et ceux obtenus avec la configuration statique optimisée. Nous insistons sur le fait que cette configuration statique optimisée est purement théorique car elle nécessite la connaissance des trafics futurs pour l'ensemble de la période étudiée, ce qui est impossible dans la réalité. De plus, la durée de la simulation considérée est relativement courte (250 min), ce qui signifie que les demandes en trafic peuvent rester « proches » de leurs valeurs moyenne sur la période. Plus la période de temps considérée pour déterminer cette configuration optimisée s'allongera, plus la probabilité que la matrice de trafic réelle dévie significativement de la matrice de trafic moyenne augmentera, et moins la configuration statique optimisée restera intéressante par rapport à une reconfiguration dynamique.

Ces valeurs ne sont que des moyennes sur les durées de simulations, et il est donc intéressant d'observer les variations réelles de la fonction coût au cours d'une simulation. Nous illustrons pour cela l'évolution temporelle des coûts sur deux exemples pour la

fonction coût M/M/1 dans les Figures 5.6 et 5.7.

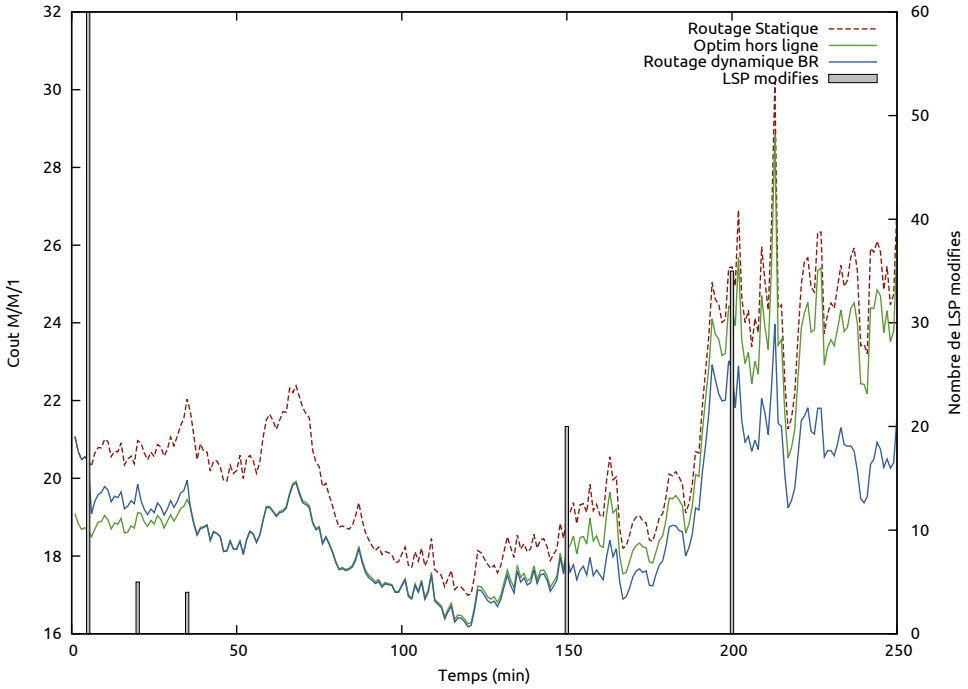


FIGURE 5.6 – Optimisation en ligne sur topologie ARPANET avec fonction coût de type M/M/1.

L'observation de ces graphiques nous révèle la dynamique suivie par le coût de la configuration dynamique (courbe bleue) et les modifications proposées et appliquées par l'algorithme Best Response (barres grises sur les graphiques). Sur toutes les topologies testées, l'algorithme commence par proposer, dès sa première exécution, un certain nombre de modifications, permettant une certaine réduction du coût (la configuration statique étant uniquement basée sur l'utilisation de plus courts-chemins). L'exécution périodique de l'algorithme lui permet par la suite de suivre l'évolution du trafic, en proposant ponctuellement des modifications permettant d'améliorer la fonction coût. On remarque sur l'exemple de la topologie ARPANET que les augmentations brutales et imprévisibles du coût sur le réseau entraînent des réactions de l'algorithme. On observe également la même tendance sur tous les scénarios : la configuration optimisée hors ligne prodigue une solution meilleure lors des premiers instants, mais la configuration dynamique devient plus intéressante au fil du temps et des évolutions du trafic.

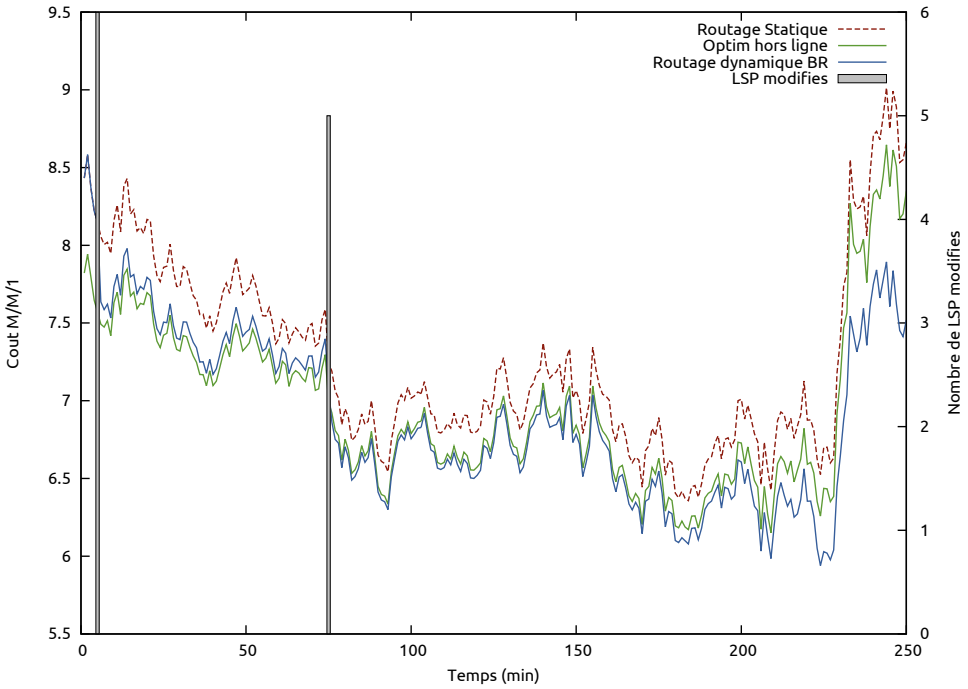


FIGURE 5.7 – Optimisation en ligne sur topologie METRO avec fonction coût de type M/M/1.

5.6.2 Modifications appliquées

Les nombres de modifications appliquées pour chaque simulation sont exposés dans les Tables 5.8 et 5.9. La première colonne présente le nombre d'instants où une reconfiguration a été proposée et appliqué par l'algorithme dynamique. Nous présentons trois valeurs pour appréhender correctement le nombre de LSP modifiés sur chaque simulation :

— **Cumul total** : la somme totale des LSP modifiées au cours d'une simulation. Autrement dit le nombre total cumulé de LSP dont la route a été touchée au cours d'une simulation de 250 minutes, il est donc possible d'y retrouver plusieurs fois un même LSP.
— **1ère reconfiguration** : le nombre de LSP distincts qui ont été modifiés lors de la toute première reconfiguration proposée par l'algorithme (la plupart du temps dès la première exécution à t=5 minutes).
— **Cumul restant** : la somme du nombre de LSP modifiés après la 1ère reconfiguration (on peut à nouveau y retrouver plusieurs fois un même LSP).

On remarque que le nombre d'instants de modifications reste modéré, et ce pour les deux fonctions coût : sur l'intégralité des simulations, au maximum 5 reconfigurations distinctes auront été nécessaires pour des simulations de 250 minutes.

Le nombre cumulé de LSP re-routés est plus ou moins proportionnel à la taille du scénario considéré. On remarque qu'une portion importante des modifications appliquées le sont dès la première reconfiguration : cela s'explique par le fait que l'on démarre avec une configuration statique de type OSPF qui n'est absolument pas optimisée. La première reconfiguration procède donc à un replacement général et poussé des LSP.

Les modifications appliquées après la première reconfiguration (« cumul restant ») sont en nombre bien moins important, ce qui indique que l'algorithme tente de se limiter aux reconfigurations minimales permettant de suivre efficacement les variations de trafic. Malgré tout, certaines valeurs restent parfois élevées (ABOVENET, ARPANET et EON pour la fonction coût quadratique). Cela nous laisse penser qu'un travail supplémentaire pourrait s'avérer nécessaire pour restreindre plus intelligemment le nombre de modifications proposées.

Topologie	**Instants de modifications**	**Nombre de LSP modifiés**		
		Cumul Total	**1ère reconfiguration**	**Cumul restant**
ABOVENET	5	92	43	49
ARPANET	3	112	46	66
BHVAC	2	52	21	31
EON	4	62	14	48
METRO	5	18	4	14
PACBELL	4	54	28	26
NSF	2	4	2	2
VNSL	2	8	5	3

TABLE 5.8 – Modifications appliquées avec la fonction Quadratique.

5.6.3 Temps d'exécution

Les temps d'exécution sont faibles, et sont impactés positivement par la restriction aux gains supérieurs à G_{Seuil} : on constate alors une valeur maximale (tous scénarios confondus) de 3 secondes pour l'exécution de l'algorithme. Ces temps sont très clairement compatibles avec une utilisation en ligne.

Topologie	Instants de modifications	Nombre de LSP modifiés		
		Cumul Total	1ère reconfiguration	Cumul restant
ABOVENET	2	40	31	9
ARPANET	5	124	60	64
BHVAC	3	36	12	24
EON	2	41	21	20
METRO	2	35	29	6
PACBELL	2	35	29	6
NSF	2	4	2	2
VNSL	2	7	5	2

TABLE 5.9 – Modifications appliquées avec la fonction M/M/1.

5.7 Conclusion

Les résultats numériques obtenus avec l'algorithme de meilleure réponse que nous proposons sont intéressants : l'erreur relative à la solution optimale reste relativement modeste (équivalente aux autres techniques testées), tout en proposant des temps d'exécution substantiellement inférieurs aux autres. Ces résultats nous font penser que cet algorithme devrait être considéré pour les systèmes de grande échelle, là où les autres techniques peuvent potentiellement rencontrer leurs limites, et là où les temps d'exécutions sont primordiaux. L'optimisation dynamique du placement est typiquement un cas où les temps d'exécutions sont importants, et les simulations effectuées en ce sens nous révèlent que l'algorithme proposé semble conserver un bon comportement dans ce type de situation. La restriction du nombre de modifications proposée par l'algorithme est une piste d'amélioration éventuelle de l'algorithme proposé pour le schéma d'optimisation dynamique.

6

Réseau overlay auto-guérissant et auto-optimisant

6.1 Introduction

Nous avons présenté dans les chapitres précédents des méthodes d'optimisation pour la reconfiguration dynamique du routage IP/MPLS, permettant d'adapter en temps réel les routes utilisées par les flux aux conditions de trafic dans le réseau. Ces méthodes peuvent être utilisées par les opérateurs de réseaux pour optimiser la performance des flux dans leurs infrastructures. Nous présentons dans ce chapitre une solution logicielle permettant le déploiement d'un réseau overlay auto-guérissant et auto-optimisant entre différents sites. La solution, directement utilisable par un fournisseur de service, est conçue pour ne nécessiter aucun changement des applications. En mesurant régulièrement la qualité des liens Internet entre les sites distants, elle permet de détecter rapidement la panne d'une route IP et de basculer le trafic sur un chemin de secours. Elle permet également de découvrir dynamiquement les chemins dans le réseau overlay qui optimisent une métrique de routage spécifique à l'application.

Cette solution logicielle a été développée dans la cadre du projet européen Panacea[1]. Ce projet regroupe des partenaires industriels et académiques collaborant pour concevoir une solution de gestion autonomique et proactive des services informatiques déployés dans le cloud afin d'en améliorer notablement la disponibilité et les performances. La solution visée est proactive dans le sens où elle mesure en permanence un

1. Proactive Autonomic Management of Cloud Resources (http ://panacea-cloud.eu)

certain nombre de paramètres et peut, en utilisant des techniques avancées d'apprentissage, prédire le temps restant avant l'interruption ou une dégradation non acceptable du service fourni. La solution visée est également autonomique dans le sens où les services déployés avec elle auront la capacité de s'auto-réparer, de s'auto-configurer et de s'auto-optimiser. Le réseau overlay présenté dans ce chapitre est un des composants innovants de cette solution et doit permettre de protéger les communications inter-clouds des dysfonctionnements de l'Internet.

Nous commençons par expliquer les motivations de ce travail en présentant des résultats expérimentaux illustrant quelques limites du routage Internet au paragraphe 6.2. Nous décrivons ensuite au paragraphe 6.3 les objectifs du système proposé et analysons les similarités et différences avec quelques solutions existantes. Le paragraphe 6.4 présente l'architecture du système et ses composants. Nous présentons en détail les choix techniques pour l'acheminement des paquets (paragraphe 6.5), pour la mesure de la qualité des chemins Internet entre les nœuds de l'overlay (paragraphe 6.6) et pour découvrir les chemins optimaux dans l'overlay avec un minimum de mesures (paragraphe 6.7). Le paragraphe 6.8 présente les plateformes de test et de validation. Enfin, le paragraphe 6.9 est consacré aux résultats expérimentaux.

6.2 Les limites du routage Internet

Il est aujourd'hui largement admis que les performances des flux pourraient être significativement améliorées en choisissant d'autres routes que celles proposées par les protocoles de routage IP [101]. Par ailleurs, si le protocole BGP a fait preuve de sa scalabilité pour connecter les 50000 AS de l'Internet [112], un certain nombre d'études ont montré qu'il pouvait être très lent à réagir et à rétablir la connectivité en cas de panne [68, 35, 60] et introduire des oscillations du plan de routage [67, 43]. Ces études ont révélé qu'un pourcentage significatif des routes IP peuvent ne pas être disponibles pendant plusieurs minutes, voire plusieurs dizaines de minutes, empêchant ainsi toute communication [67, 91]. Depuis une quinzaine d'années, on sait ainsi que le routage de l'Internet souffre de nombreux problèmes en termes de disponibilité et d'optimalité des chemins proposés.

Nous décrivons ci-dessous les résultats d'une expérience qui montrent que la situation n'a pas vraiment évolué depuis que ces études ont été réalisées. Pour réaliser cette expérience, nous avons sélectionné 20 nœuds du réseau NLNog ring [2] (cf. Figure 6.1). Nous avons mesuré la latence de communication et le taux de perte entre chaque paire de machines toutes les deux minutes pendant une semaine. La mesure est effectuée en envoyant 5 paquets consécutifs avec l'utilitaire "ping" basé sur le protocole Internet Control Message Protocol (ICMP). Lorsque les 5 paquets sont perdus, nous considérons

2. Ce réseau est constitué de 293 machines réparties dans 46 pays (cf. https ://ring.nlnog.net)

que la route IP entre la source et la destination n'est pas disponible.

FIGURE 6.1 – Localisation géographique des nœuds utilisés.

Les observations principales sont les suivantes :

— La route IP n'est pas disponible au moins une fois dans la semaine pour 65% des couples origine/destination. Dans 21% des cas, la route n'est pas disponible durant plus de 4 minutes (et même plus de 14 minutes pour 11% des cas).

— La latence des routes IP exhibe des variations brutales et imprévisibles (cf. Figure 6.2), avec parfois des variations de plus de 500%. Il ne semble pas qu'un modèle simple, par exemple markovien, puisse représenter correctement l'évolution de la latence de chaque lien.

— La route IP n'est pas la route à latence minimale dans 38% des cas. A chaque instant de mesure, il y a au moins un couple origine/destination pour lequel la latence pourrait être réduite de plus de 76% en choisissant une autre route que celle calculée par les protocoles de routage Internet.

— Plus de 11% des routes IP ont un taux de perte supérieur à 1%. En choisissant d'autres chemins que ceux de l'Internet, on aurait pu obtenir un taux de perte proche de 0%.

Ces résultats confirment les études mentionnées précédemment. Les pertes de connectivité se produisent plus fréquemment que ce que l'on ne pourrait croire dans l'Internet, et, de manière générale, les chemins utilisés sont assez souvent sous-optimaux. Soulignons toutefois que les performances observées entre machines situées dans des pays occidentaux sont significativement meilleures.

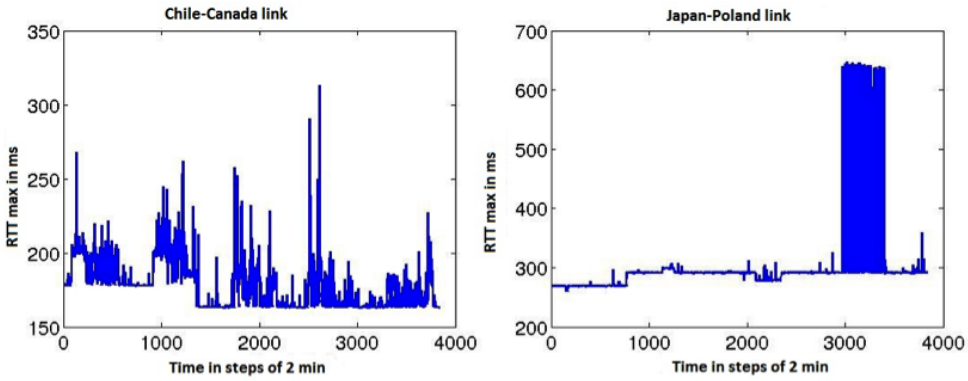


FIGURE 6.2 – Variation du RTT des routes IP Chili-Canada et Japon-Pologne.

6.3 Un réseau overlay auto-guérissant et auto-optimisant

Nous avons vu au paragraphe 6.2 que le routage Internet est d'une part sous-optimal et d'autre part relativement long à rétablir la connectivité du réseau en cas de panne. Si le service fourni par l'Internet est suffisant pour la plupart des applications, certaines ont besoin d'un niveau de disponibilité et de qualité de service supérieur. Malheureusement, l'ossification de l'Internet ne permet pas de réaliser des changements pourtant unanimement reconnus comme nécessaires. Pour adapter dynamiquement les routes utilisées à la demande en trafic, les opérateurs ont la possibilité d'utiliser les techniques proposées dans les chapitres précédents. Une autre approche, permettant aux utilisateurs de contrôler la façon dont leur flux de données sont routés dans l'Internet, est cependant possible et repose sur l'utilisation des réseaux overlays. Nous décrivons au paragraphe 6.3.1 le principe de cette approche.

Dans le cadre du projet FP7 PANACEA, nous avons développé un prototype de système de communication innovant basé sur cette approche. Ce système est en fait un réseau overlay auto-guérissant et auto-optimisant capable d'optimiser le routage des flux en fonction des besoins applicatifs. Nous présentons au paragraphe 6.3.2 les exigences fonctionnelles auxquelles le système proposé cherche à répondre. Enfin, nous expliquons au paragraphe 6.3.3 les similarités et différences de notre système avec d'autres solutions basées sur l'utilisation de réseaux overlays.

6.3.1 Les réseaux overlays

L'utilisation des réseaux overlay a été proposée comme une solution fournissant aux applications la flexibilité et le contrôle souhaités en termes de routage de leurs propres flux [93, 113, 42, 20]. Comme illustré dans la Figure 6.3, un réseau overlay correspond

à un réseau virtuel de niveau applicatif déployé entre un certain nombre de machines au-dessus d'une infrastructure IP. Les liens du réseau overlay correspondent alors à des chemins dans le réseau sous-jacent, ces chemins étant calculés par les protocoles de routage Internet. Les nœuds du réseau overlay ont toutefois la possibilité de coopérer entre eux pour le routage des paquets, chaque nœud pouvant servir de relais pour les paquets échangés entre deux autres nœuds.

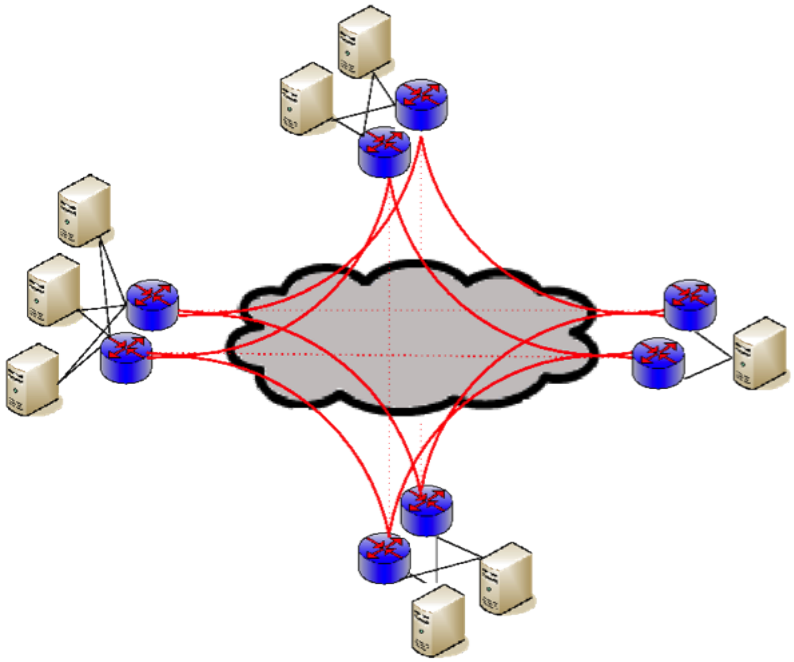


FIGURE 6.3 – Structure d'un réseau overlay.

L'avantage de cette approche est qu'il est possible de tirer parti de la redondance de la topologie en termes de routes. Ainsi, les nœuds de l'overlay ont la possibilité de détourner "à la volée" la route d'un paquet pour suivre un autre chemin lorsque le chemin primaire n'est plus disponible ou qu'une dégradation significative de ses performances est observée. De même, en utilisant ce principe, les nœuds de l'overlay sont capables de choisir les chemins entre eux ayant les meilleures performances.

6.3.2 Objectifs de la solution proposée

Le système de communication que nous proposons est en fait un réseau overlay formé par des routeurs logiciels déployés en différents endroits de l'Internet. Ces routeurs sont capables de mesurer la qualité des chemins Internet les reliant (grâce à l'envoi de paquets sondes) et ainsi, en utilisant des techniques d'apprentissage, de découvrir les chemins

optimaux entre eux dans le réseau overlay. Outre des performances optimisées, l'objectif est d'avoir un très haut niveau de disponibilité du service de communication en détectant très rapidement la disparition de la route IP connectant deux nœuds et en basculant le trafic sur un chemin de secours. Plus précisément, le système de communication que nous proposons cherche à répondre aux exigences fonctionnelles suivantes :

- Le système doit être *auto-guérissant* dans le sens où il doit être capable de détecter rapidement le dysfonctionnement d'un chemin et de basculer le trafic sur un chemin de secours.
- Le système doit être *auto-optimisant* dans le sens où il doit être capable de découvrir en permanence les routes optimales dans le réseau overlay.
- Le système doit pouvoir *passer à l'échelle* pour être utilisé dans des réseaux overlays de plusieurs centaines de nœuds, ce qui implique l'utilisation de méthodes efficaces pour apprendre les chemins optimaux avec une quantité de mesures minimale.
- Le système doit pouvoir intégrer les *métriques et les contraintes de routage propres à une application*, permettant ainsi de répondre aux besoins spécifiques de chaque application.
- Le système doit pouvoir fonctionner sans *aucun changement applicatif ou du système d'exploitation*, de telle sorte qu'il est possible de contrôler le routage des flux d'applications du commerce.

Nous allons détailler dans les paragraphes suivants l'architecture et les solutions techniques que nous proposons pour aboutir à un système conforme à ces exigences. Avant cela, nous détaillons dans le prochain paragraphe les similarités et différences de notre système avec d'autres solutions basées sur les réseaux overlays.

6.3.3 Similarités et différences par rapport aux solutions existantes

Les réseaux overlays ont été utilisés par le passé pour résoudre des problèmes dans différents domaines, avec des objectifs très différents. Pour ne citer que quelques exemples, ils ont permis l'auto-organisation dans les réseaux pair-à-pair, notamment avec les systèmes Chord [109], Pastry [100] et Tapestry [123], l'implémentation du multicast au niveau applicatif [30, 19, 92, 70], ou encore la protection contre les attaques DDos (Distributed Denial of Service) [110, 118]. Les réseaux overlays sont également à la base de la solution d'Akamai Tech Inc pour la distribution dynamique de contenus [15, 98, 69, 85, 105].

Aujourd'hui, une des utilisations importantes des réseaux overlays consiste à fournir des réseaux de niveau 2 étendus au dessus d'infrastructures IP de niveau 3. Ainsi la technologie des « Virtual eXtensible LANs » (VXLAN) résout le problème de l'extension géographique des VLAN [3] [79]. Un VXLAN permet d'une part de constituer un VLAN

3. La technologie des Virtual LANs (VLAN) permet de faire cohabiter plusieurs réseaux logiques

au dessus d'Internet entre des machines IP géographiquement distantes et augmente le nombre de VLAN de 4096 à plus de 16 millions, un nombre jugé suffisant pour satisfaire les besoins des centres de données dans les années à venir. Une des limitations du déploiement à grande échelle des VXLAN est la nécessité d'avoir le multicast activé sur les réseaux concernés. Pour contourner ce problème, le protocole "Distributed Overlay Virtual Ethernet" (DOVE) améliore les VXLAN en fournissant son propre plan de commande afin d'éviter la limitation de multicast [5]. La différence majeure du système de communication que nous proposons avec des technologies comme les VXLAN ou DOVE réside dans les propriétés auto-guérissante et auto-optimisante du système, alors que ces technologies s'appuient sur les routes calculées par les protocoles de routage Internet sans chercher à contrôler le routage des flux.

De ce point de vue, nos travaux sont beaucoup plus proches des systèmes développés dans le cadre des projets Detour et RON. Detour [33] est un framework logiciel permettant la mise en place d'un réseau overlay pour améliorer le routage Internet. Grâce à des modifications du noyau du système d'exploitation (en l'occurrence, FreeBSD), les paquets sont encapsulés et routés sur des chemins alternatifs calculés en fonction d'opérations de mesures et de surveillance du réseau. Si l'implémentation au niveau du noyau s'avère très efficace, elle a représenté un inconvénient majeur de la solution qui a freiné son adoption par la communauté. La solution proposée par le projet Resilient Overlay Network (RON) [14] est une bibliothèque logicielle permettant de créer un réseau overlay entre les applications qui l'utilisent. En procédant à une surveillance permanente de la qualité de tous les liens de l'overlay, les nœuds peuvent réagir rapidement aux pannes et découvrir les chemins optimaux entre eux. Par rapport aux solutions proposées par Detour ou RON, si notre système a les mêmes objectifs, il a l'avantage de pouvoir fonctionner avec des applications du commerce sans aucune modification ni de l'application ni du système d'exploitation. D'autre part, Detour et RON reposent sur l'évaluation de la qualité de tous les liens de l'overlay, ce qui les limite évidemment à des réseaux de petite taille, là où notre système a été conçu pour découvrir les chemins optimaux avec un effort de monitoring minimal.

Ce dernier objectif constitue un point commun avec l'approche de routage utilisée dans les *Cognitive Packet Networks* (CPN) [55, 54]. Les CPN utilisent un routage guidé par la qualité de service et s'auto-optimisent de manière distribuée en apprenant les routes optimales grâce à des paquets spéciaux (les "smart packets") qui mesurent la qualité de service sur les routes qu'ils empruntent. Les décisions de routage sont prises par chaque nœud du réseau et sont basées sur des techniques d'apprentissage à base de réseaux neuronaux. Les CPN se sont montrés extrêmement efficaces dans une grande variété de contextes applicatifs [4] et ont de nombreuses similitudes avec le système que nous proposons : les deux approches visent une solution de routage auto-guérissante et

virtuels isolés les uns des autres sur des équipements réseau de niveau 2.

4. cf. http ://san.ee.ic.ac.uk/publications.shtml pour une liste complète de références

auto-optimisante passant à l'échelle. Cependant, notre système est spécifiquement conçu pour les réseaux overlays, alors que l'approche CPN fonctionne typiquement au niveau de la couche réseau (même si l'application des techniques des CPN a été envisagée pour des réseaux overlays pair-à-pair dans [56]). Une autre différence majeure entre les deux approches est liée au fait que nous nous intéressons à des approches d'apprentissage fournissant des garanties de performance pire-cas basées sur une généralisation du problème du bandit manchot [5]. Enfin, alors que les technologies CPN font l'objet d'un brevet [53], le logiciel que nous avons développé devrait être publié en tant que logiciel libre.

Pour conclure, nous notons qu'il y a actuellement un effort de recherche important sur les réseaux définis par le logiciel avec notamment l'approche "Software-Defined Networking" (SDN) dont l'objectif est de découpler le plan de contrôle du réseau de son plan de données, afin de rendre le réseau directement programmable par les applications [4]. Cette approche va à termes offrir une granularité de contrôle du réseau beaucoup plus fine que l'approche overlay que nous considérons, puisque les possibilités de contrôle ne se limitent pas aux extrémités du réseau. Il faut toutefois noter que les réseaux SDN ne sont utilisés à l'heure actuelle que dans les centres de données avec des outils comme Open vSwitch [58]. Leur utilisation par des opérateurs de télécommunications dans des réseaux WAN est encore un sujet de recherche.

6.4 Architecture du réseau overlay

Dans cette partie, nous présentons l'architecture proposée pour atteindre les objectifs présentés au paragraphe 6.3.2. Nous fournissons tout d'abord une vue globale de l'architecture au paragraphe 6.4.1, puis présentons ses composants au paragraphe 6.4.2.

6.4.1 Vue globale de l'architecture

La solution proposée consiste à définir un réseau overlay sur lequel transitent les données échangées entre différents sites [6]. Dans chaque site de l'overlay, on trouve des machines clientes exécutant des agents logiciels. Pour chaque machine cliente, deux agents logiciels sont chargés de fournir les points d'entrée et de sortie du réseau overlay : l'agent de transmission et l'agent de réception. À chaque site est associé un proxy prenant en charge la mesure de la qualité des liens vers certaines destinations, la sélection des chemins à utiliser pour ces destinations ainsi que le transfert des paquets émis vers ces destinations. Cette architecture est visible sur la Figure 6.4.

Dans l'exemple de la Figure 6.4, le lien Internet direct entre les sites 1 et 2 subit une coupure. Cette dernière est détectée par le système de mesure intégré aux proxies,

5. La comparaison avec les techniques d'apprentissage utilisées dans les CPN est toutefois une perspective de recherche intéressante.

6. Dans le contexte de Panacea, les sites correspondent à des plateformes de cloud computing.

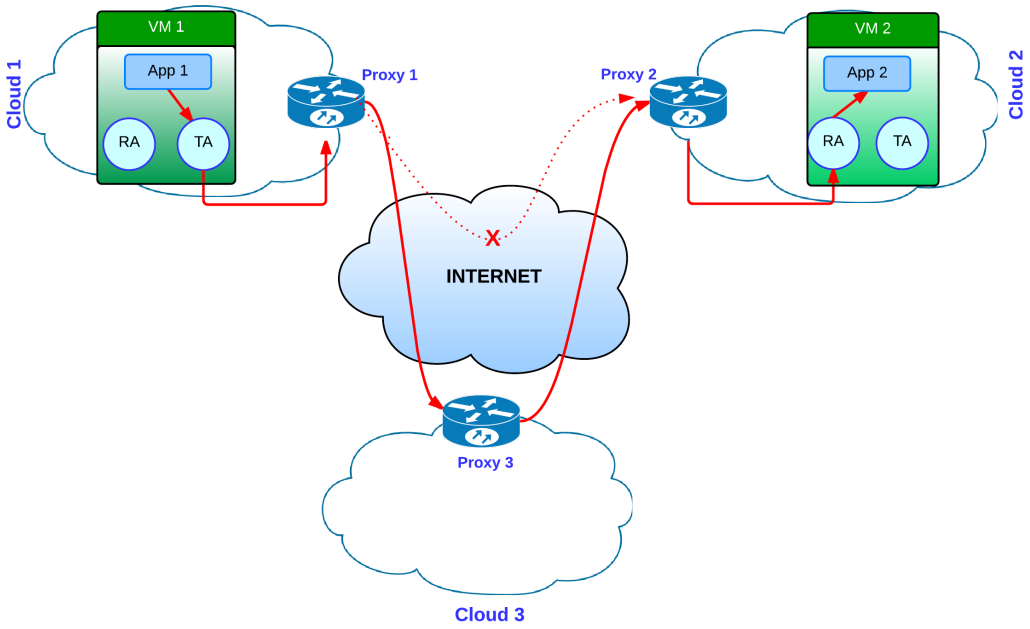


FIGURE 6.4 – Architecture globale du réseaux overlay avec ses différents agents.

qui décident alors de rediriger le trafic vers un proxy intermédiaire, assurant ainsi la continuité du service.

6.4.2 Les composants de l'architecture

6.4.2.1 Les agents de transmission et de réception

Rappelons qu'un de nos objectifs essentiels est de pouvoir contrôler le routage des flux applicatifs sans qu'aucune modification de l'application ne soit nécessaire, c'est-à-dire sans que l'application n'ait même conscience que ses données sont routées à travers le réseau overlay. Pour cela, nous proposons d'utiliser un mécanisme d'interception et d'encapsulation des paquets, opérant de façon transparente pour l'application. Ces opérations sont réalisées grâce à deux agents logiciels s'exécutant sur les machines émettrice et réceptrice :

— **Agent de transmission** : les paquets envoyés par l'application vers une destination distante faisant partie du réseau overlay sont interceptés de façon transparente par l'agent de transmission (TA) local, encapsulés par lui dans d'autres paquets IP (encapsulation IP-in-IP) qui sont ensuite transmis au proxy du site source. Ce dernier se chargera alors du transfert des paquets vers le site destina-

tion.

— **Agent de réception** : l'encapsulation des paquets mise en œuvre à l'entrée de l'overlay implique nécessairement l'existence d'une autre entité chargée de leur désencapsulation à la sortie de l'overlay. L'agent de réception (RA) est le programme chargé de l'extraction du paquet originel et de sa livraison à l'application destination qui s'exécute sur la même machine que lui.

Les choix techniques retenus pour l'interception et l'encapsulation seront détaillés dans les paragraphes 6.5.1 et 6.5.2. Nous détaillerons également au paragraphe 6.5.4 certaines opérations nécessaires pour que les paquets soient bien livrés à leur destination.

6.4.2.2 Le proxy

Le proxy est un routeur logiciel prenant en charge l'acheminement des paquets vers leur destination. Si dans l'implémentation actuelle il n'y a qu'un seul proxy dans chaque site, il n'y a pas *a priori* de difficultés majeures à l'utilisation de plusieurs proxies pour équilibrer la charge. Comme illustré sur la Figure 6.5, le proxy est en fait une entité constituée de trois agents :

— **Agent de mesure** : cet agent implémente les méthodes permettant de mesurer la latence, le taux de perte ou la bande-passante disponible sur un chemin dans l'overlay. Il peut être consulté par l'agent de routage pour estimer la qualité d'un chemin donné suivant une certaine métrique. Il peut également être configuré pour surveiller à intervalles réguliers la disponibilité d'un chemin. Nous décrivons plus en détails les méthodes utilisées pour mesurer la qualité des liens de l'overlay au paragraphe 6.6.

— **Agent de routage** : cet agent est configuré pour optimiser les métriques de routage propres à une application. Pour cela, il pilote l'agent de mesure de manière à découvrir un chemin optimal avec un effort de mesure minimal. Nous détaillons au paragraphe 6.7 les méthodes utilisées pour découvrir un chemin optimal. C'est le chemin découvert par l'agent de routage qui sera inscrit dans la table de routage de l'agent d'acheminement des paquets.

— **Agent d'acheminement des paquets** : cet agent est chargé du transfert des paquets sur un chemin de bout-en-bout entre la source et la destination. Sa table de routage détermine ici le chemin complet qui sera emprunté pour atteindre chaque destination : il s'agit d'un routage à la source, où l'intégralité du chemin est défini dès son point de départ (par opposition à un routage de proche en proche).

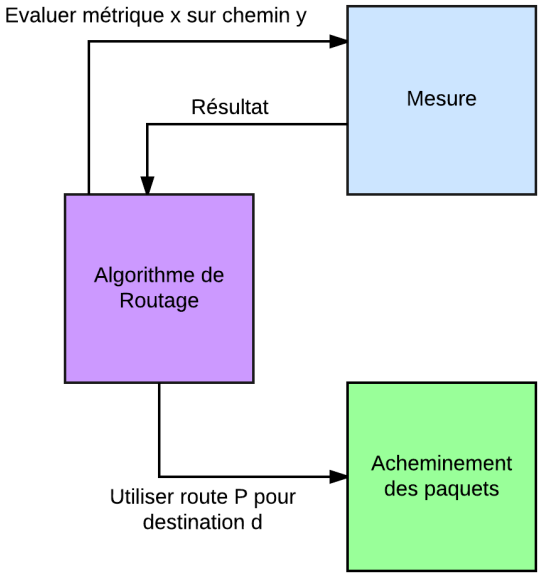


FIGURE 6.5 – Interactions entre les entités constituant le proxy.

6.5 Interception, encapsulation et acheminement des paquets

La Figure 6.6 illustre le processus de transmission d'un paquet dans le réseau overlay. Quand un paquet est envoyé par une tâche vers un réseau distant, l'agent de transmission intercepte le paquet puis l'encapsule dans un autre paquet IP qui est envoyé au proxy local. Ce dernier examine sa table de routage pour déterminer le chemin de bout-en-bout dans le réseau overlay à utiliser pour joindre la destination. Chaque proxy intermédiaire transmet la paquet au suivant jusqu'à la destination. Le proxy du réseau destination transmet enfin le paquet à l'agent de réception de la machine destinatrice, qui peut livrer le paquet originel à sa destination. Nous présentons ci-dessous les détails techniques de chacune de ces opérations.

6.5.1 Interception des paquets

L'agent de transmission doit disposer d'un moyen pour intercepter les paquets destinés à une machine distante du réseau overlay. On ne souhaite pas intercepter les paquets à destination d'une machine ne faisant pas partie du réseau overlay. Pour cela, nous utilisons une solution directement basée sur les mécanismes d'interception des systèmes d'exploitation. Le principe est le suivant : mise en place de règles de filtrage au niveau

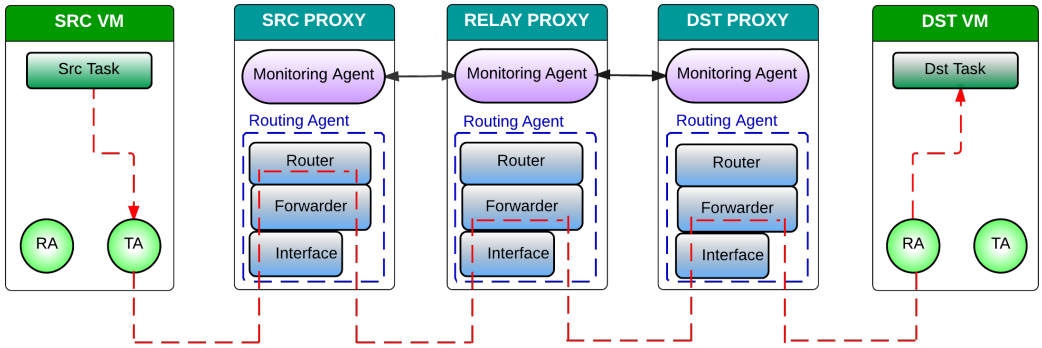


FIGURE 6.6 – Interception, encapsulation et acheminement des paquets.

du pare-feu de la machine hôte permettant de faire remonter le paquet vers l'espace utilisateur en l'envoyant vers une interface réseau spécifique, utilisable par les programmes qui le souhaitent [7].

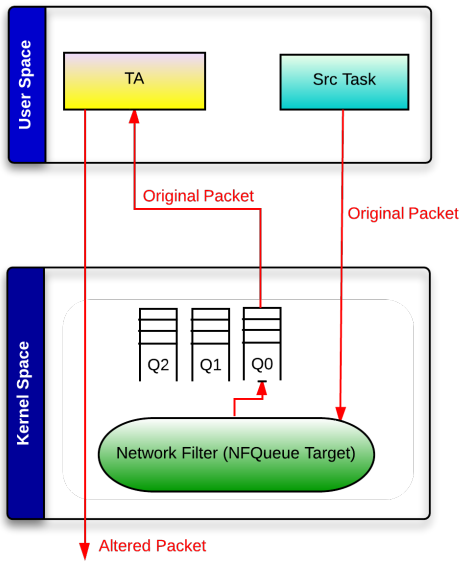


FIGURE 6.7 – Mécanisme d'interception sur Linux avec la Network Filter Queue.

7. Le système d'interception des paquets étant spécifique à chaque système d'exploitation, il s'agit de la seule partie non portable du code développé. Au moment où ce mémoire est rédigé, l'implémentation a été faite uniquement pour Linux.

Le principe global de l'interception sur Linux est présenté sur la Figure 6.7. Sur ce système d'exploitation, le pare-feu Netfilter [8], permet aisément de mettre en place ce type d'interception en déclarant des règles de filtrage via l'outil iptables. Ce dernier permet de filtrer les paquets correspondant à certains critères (IP source, IP destination, protocole, port, ...), et de les transférer vers une cible spécifique nommée Network Filter Queue (NFQueue). Cette dernière est accessible aux programmes qui souhaitent l'utiliser pour lire les paquets, tels que notre agent de transmission. Ainsi l'agent de transmission que nous avons développé est chargé d'écrire dynamiquement les règles de filtrage pour intercepter les paquets à destination de machines appartenant au réseau overlay. Il récupère alors tous les paquets interceptés dans la file d'attente NFQueue, avant de les encapsuler et de les transmettre au proxy.

Remarque 6.5.1. *Nous désactivons l'interception lorsque le meilleur chemin déterminé par le proxy correspond à la route IP directe. Le proxy informe régulièrement l'agent de transmission des destinations pour lesquelles il doit intercepter les paquets.*

Remarque 6.5.2. *L'interception des paquets sur la NFQueue ne fonctionne pas dans l'émulateur CORE (cf. paragraphe 6.8.1) sans le patch fourni sur le site internet du projet CORE.*

6.5.2 Encapsulation

Une fois interceptés par l'agent de transmission, les paquets doivent être transmis au proxy local en étant préalablement encapsulés. La Figure 6.8 illustre le processus d'encapsulation.

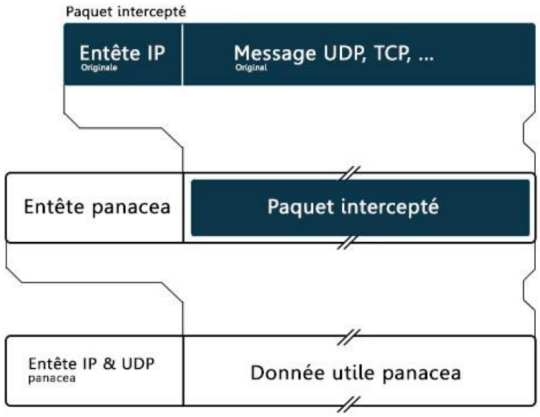


FIGURE 6.8 – Encapsulation des paquets sur l'overlay Panacea.

À partir de chaque paquet intercepté, nous construisons un paquet "Panacea" en ajoutant une entête particulière, appelée entête Panacea, qui va nous servir par la suite

pour le routage des paquets via le réseau overlay. La structure de cette entête est présentée sur la Figure 6.9. Elle contient notamment l'adresse IP du proxy destination et la route à suivre dans le réseau overlay pour l'atteindre (séquence de proxies intermédiaires). Au niveau de l'agent de transmission, cette route n'est pas encore connue et ce champs est donc laissé vide[8]. Finalement, le paquet Panacea est transmis au proxy local en utilisant UDP.

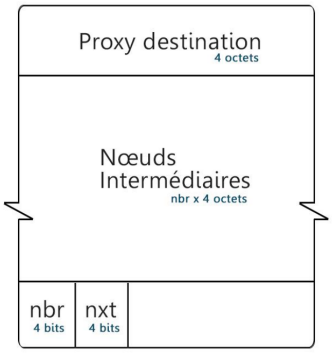


FIGURE 6.9 – Structure de l'entête Panacea.

6.5.3 Traitement par l'agent d'acheminement du proxy

L'agent d'acheminement du proxy permet d'acheminer le paquet de données jusqu'à sa destination finale. Lorsqu'il reçoit un paquet Panacea, l'agent inspecte l'entête Panacea pour déterminer quel est son rôle précis. Trois cas peuvent se présenter :

- Le paquet est au niveau du proxy source : c'est le cas si le proxy n'est pas la destination et que le champ décrivant la route à suivre est vide. L'agent utilise alors sa table de routage pour écrire le chemin complet à utiliser dans l'entête Panacea du paquet encapsulé (voir Figure 6.9), puis envoie le paquet vers le prochain proxy du chemin.
- Le paquet est au niveau d'un proxy intermédiaire : l'agent se contente de transmettre le paquet vers le prochain nœud, en mettant à jour l'IP destination pour qu'elle contienne le prochain proxy.
- Le paquet est arrivé au niveau du proxy destination : l'agent transfère le paquet vers l'agent de réception de la machine destination.

Nous présentons le fonctionnement de l'agent d'acheminement sur la Figure 6.10. Il se compose d'une interface réseau, d'une entité de transfert et d'un routeur logiciel. L'interface réseau UDP fonctionnant sur un port particulier commun à tous les proxies permet de récupérer le paquet de données. L'entité de transfert décide soit de le faire

8. La route complète sera déterminée et inscrite dans l'entête du paquet Panacea par le proxy local.

passer au routeur logiciel, soit de l'envoyer vers sa prochaine destination si le paquet est en transit, soit de le livrer à sa destination finale.

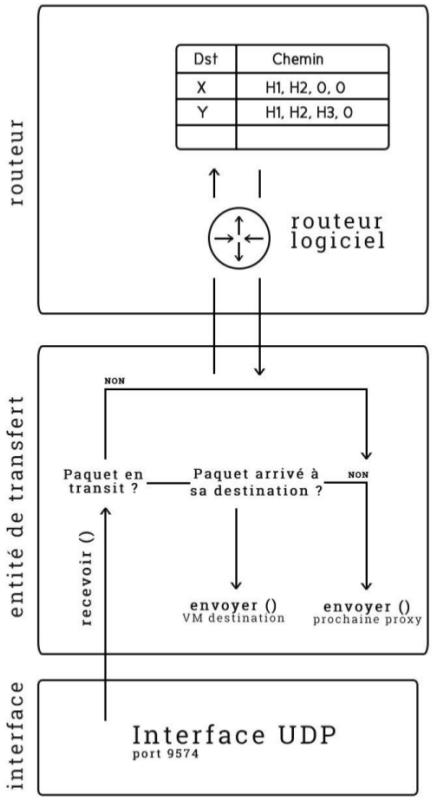


FIGURE 6.10 – Fonctionnement de l'agent d'acheminement.

6.5.4 Désencapsulation et transmission finale des paquets

L'agent de réception permet de désencapsuler le paquet originel et de le livrer à l'application réceptrice s'exécutant sur la même machine que lui. Pour cette application, tout doit se passer comme si le paquet venait directement d'Internet. À cet effet, nous utilisons une interface réseau particulière appelée "raw socket". Les raw sockets offrent un accès direct à la couche 3 du modèle OSI, et permettent de construire "manuellement" un paquet. Les raw sockets sont un outil très puissant qui nécessitent les droits administrateur. Leur utilisation nous permet de transmettre le paquet désencapsulé en conservant les champs IP source et IP destination du paquet d'origine. Lorsque que le paquet est transmis au système d'exploitation, il est transmis immédiatement à l'application réceptrice car l'adresse IP destination correspond à celle de la machine.

Cette étape comporte une difficulté supplémentaire lorsque l'adresse IP publique de la machine est différente de son adresse privée. Les machines situées dans différents

sites communiquant via leurs adresses publiques, le paquet encapsulé contient donc des adresses IP publiques. La transformation automatique des adresses publiques en adresses privées par le mécanisme de Network Address Translation (NAT) n'est possible que pour le paquet Panacea, et non pour le paquet d'origine encapsulé. Ainsi, au niveau de l'agent de réception, l'entête du paquet d'origine contient toujours l'adresse IP publique pour la destination, et non l'adresse privée. Si l'on se contente d'envoyer le paquet d'origine ainsi sur le raw socket, il est rejeté car l'adresse est considérée comme erronée.

Pour régler ce problème nous fournissons à l'agent de transmission, via un fichier de configuration, son adresse IP publique et son adresse IP privée. En cas de différence entre les deux, il se charge alors automatiquement de modifier l'adresse IP publique en adresse IP privée. Il s'assure également de bien mettre à jour la somme de contrôle du paquet IP (niveau 3) pour qu'elle reflète ce changement et éviter que le paquet ne soit jeté à cause d'une somme erronée. Il faut également faire attention aux sommes de contrôle des protocoles transport (niveau 4) car elles prennent souvent en compte l'entête IP dans leur calcul. C'est le cas des protocoles TCP, UDP et ICMP, pour lesquels nous devons donc aussi corriger la somme de contrôle [9]. L'agent transmet enfin le paquet corrigé sur le raw socket, qui lui permet d'arriver correctement à sa destination.

6.6 Mesure de la qualité des liens du réseau overlay

L'agent de mesure implémente les méthodes permettant de mesurer la latence, le taux de perte ou la bande-passante disponible sur un chemin dans l'overlay. Nous décrivons ci-dessous les méthodes utilisées pour estimer chacune de ces métriques.

6.6.1 Mesure des latences et pertes

Le principe de mesure de la latence et des pertes est relativement simple. Nous envoyons des paquets sondes en précisant le chemin à suivre. À chaque passage par un nœud du chemin, un tampon contenant la date de passage est ajouté dans le paquet. Lorsque le paquet revient à sa source, il contient toutes les dates de passage sur le chemin, dans les deux sens. Ainsi nous pouvons facilement à la fin de l'opération déduire la latence sur chaque segment du réseau. Nous envoyons 5 paquets sondes consécutifs pour calculer le délai moyen de transmission sur chaque lien du chemin. Un paquet qui ne revient pas à sa destination avant un certain temps (fixé à 10s, mais configurable par l'utilisateur) est considéré comme perdu. On peut ainsi déterminer un pourcentage de pertes imprécis mais suffisant pour détecter la défaillance d'un chemin. La structure des paquets sonde est présentée sur la Figure 6.11.

9. Ce sont les seuls protocoles pour lesquels on fait le recalcul des sommes de contrôle de niveau 4 dans l'implémentation actuelle. Les procédures pour les autres protocoles pourront être ajoutées par la suite, en fonction des besoins.

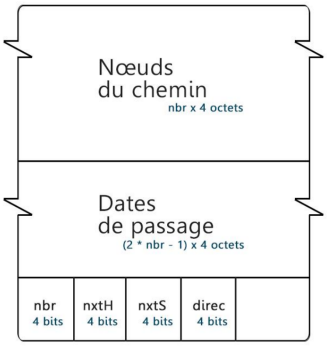


FIGURE 6.11 – Structure d'un paquet sonde pour la mesure de latence.

Cette structure est assez similaire à l'entête d'un paquet de données Panacea. Elle est constituée principalement d'un champ contenant les nœuds du chemin à évaluer et d'un autre pour les dates de passage. Les autres champs sont des compteurs utilisés pour déterminer le prochain nœud et la direction de propagation du paquet (aller ou retour). Dans notre implémentation nous fixons arbitrairement le nombre maximum de sauts pour un chemin grâce au paramètre L_{max} (par défaut fixé à la valeur 4).

6.6.2 Mesure de la bande passante

La bande passante disponible est une métrique essentielle pour certaines applications. Nous évaluons ci-dessous deux techniques d'estimation de cette métrique déjà présentées au paragraphe 2.6.3 : la méthode passive basée sur la formule SQRT et la méthode active Train Of Packet Pair (TOPP). Elles ont uniquement été évaluées et n'ont pas encore été implémentées dans le système de routage.

6.6.2.1 Estimation passive pour TCP

Nous comparons ici l'estimation fournie par la méthode SQRT avec la bande passante mesurée via l'utilitaire iperf sur l'émulateur CORE (cf. paragraphe 6.8.1). On considère un cas simple comportant deux nœuds et un lien, pour lequel nous fixons un délai et un taux de perte de paquets. Nous exécutons des mesures d'une durée de 30 secondes avec iperf. Chaque mesure est répétée 10 fois. Les résultats obtenus pour un RTT de 100ms sont présentés sur la Figure 6.12. L'erreur relative mesurée est particulièrement importante et souvent supérieure à 100%. La piètre qualité des résultats obtenus, ajouté au fait que la méthode nécessite une très bonne estimation des pertes et n'est applicable que pour le protocole TCP, nous pousse à considérer une méthode active plus évoluée, la méthode TOPP.

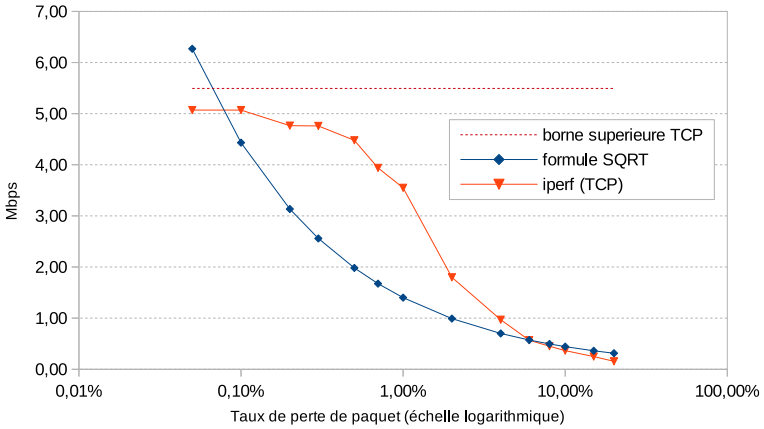


FIGURE 6.12 – Évolution de la bande passante TCP en fonction du taux de perte, avec un RTT de 100ms, W=64Ko et paquet de 1.5Ko.

6.6.2.2 Estimation active avec la méthode TOPP

Validation en émulation Pour observer le fonctionnement de TOPP, nous émulons un réseau très simple avec l'émulateur CORE, avec 2 nœuds reliés par un unique lien, de capacité fixée C. Les délais et pertes sur le lien sont fixés à 0.

Nous avons implémenté un programme simple permettant de tracer la courbe $\frac{\Delta_t}{\Delta_s}$, en fonction du débit d'envoi (cf section 2.7.2 pour la présentation précise de la méthode). Nous introduisons un trafic UDP utilisant 50% de la capacité du lien. Les courbes obtenues pour 4 capacités de liens sont tracées sur la Figure 6.13.

Pour les trois plus petites capacités, on observe globalement la tendance attendue : valeur constante environ égale à 1 pour $D < A$, puis croissance pour $D > A$. On note que le point d'inflexion de la courbe est à peu près situé au niveau de la bande passante disponible. Les choses sont plus complexes pour $C = 5000$ kbps, où il est impossible d'observer une quelconque tendance : l'envoi de 2 paquets successifs s'avère rapidement insuffisant pour qu'une variation soit visible pour des bandes passantes importantes.

On remarque qu'un certain bruit est observable pour les trois premières configurations : la méthode TOPP requiert en effet un filtrage des valeurs mesurées. Ce dernier est particulièrement complexe à mettre en œuvre. C'est pourquoi nous considérerons l'implémentation fournie par l'outil IGI [61]. Nous testons les résultats obtenus avec cet outil sur les 4 configurations précédentes, avec et sans trafic. Les résultats (moyennes sur 10 mesures) sont présentés dans la Table 6.1.

L'outil IGI fournit ici de bonnes estimations de bande passante. On note que les temps de mesures s'échelonnent de 1.6 secondes à plus de 50 secondes, suivant la bande passante disponible. Une expérimentation pour un RTT de 400 ms nous a montré que le RTT du lien ne semble pas impacter trop fortement les temps de mesure (le temps de

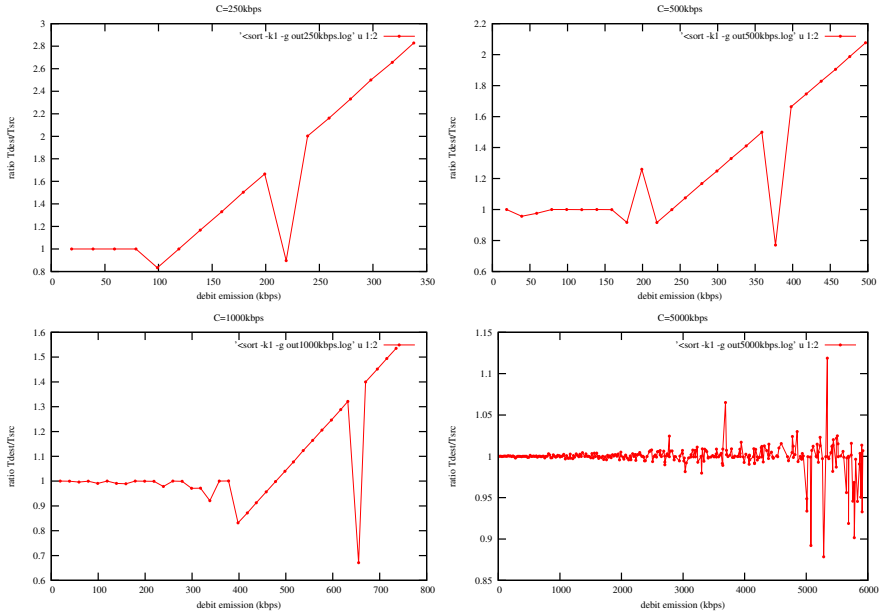


FIGURE 6.13 – Méthode TOPP : comportement pour un lien utilisé à 50%.

C (kbps)	Trafic de fond nul			Trafic de fond 50%		
	A (kbps)	IGI (kbps)	t(s)	A (kbps)	IGI (kbps)	t(s)
250	250	246	15,9	125	117	52,2
500	500	499	8,3	250	234	27,1
1000	1000	973	4,6	500	467	14,0
5000	5000	5015	1,6	2500	2302	4,5

TABLE 6.1 – Évaluation de bande passante avec l'outil IGI, avec un RTT=40ms.

mesure maximum est alors d'environ 1 minute).

Validation dans l'Internet Pour confirmer ces bons résultats, nous décidons de tester l'outil IGI pour déterminer la bande passante disponible sur un vrai chemin internet entre deux machines distantes. À nouveau, nous comparons les résultats obtenus avec ceux obtenus via l'utilitaire de mesure iperf, en insérant progressivement un trafic de fond UDP. Chaque point de mesure correspond à la moyenne de 10 mesures successives. Les estimations de bande passante obtenues sont résumées sur la Figure 6.14. Un certain pourcentage d'erreur est observable pour les trafics de fond les plus faibles, mais la précision s'améliore lorsque la capacité disponible devient plus faible. Ces résultats démontrent que la méthode TOPP est globalement fiable.

Il est aussi important de quantifier l'effort nécessaire à l'obtention des résultats. La

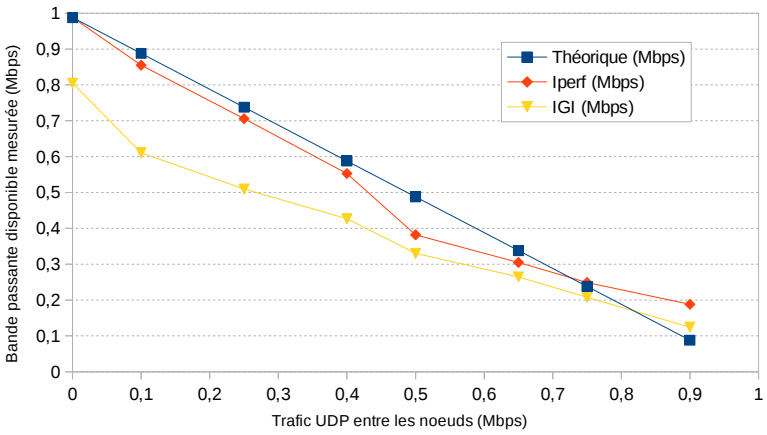


FIGURE 6.14 – Bande passante estimée par IGI sur un vrai chemin internet.

Figure 6.15 montre la quantité totale de données envoyées sur le réseau pour chaque mesure, ainsi que le débit moyen d'envoi des paquets de mesure. On constate que l'envoi de quelques centaines de ko est nécessaire pour l'obtention d'une mesure, le débit de mesure décroissant très rapidement avec la capacité disponible.

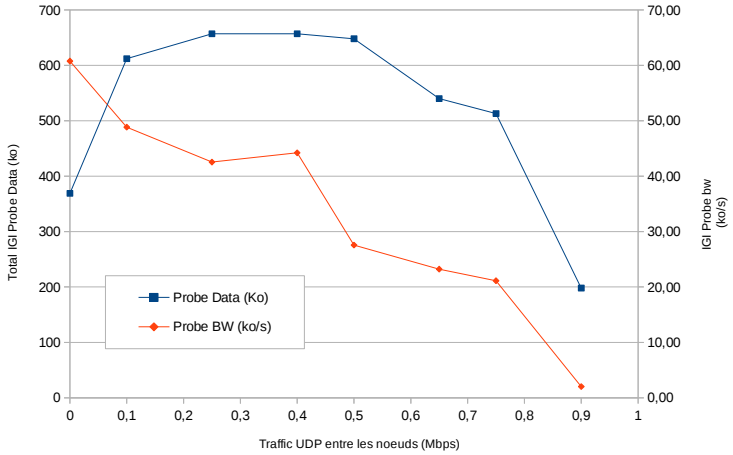


FIGURE 6.15 – Effort de mesure pour estimer la capacité disponible sur un lien Internet.

Qualité du choix des chemins Les estimations fournies par l'outil IGI subissent des variations parfois non négligeables. Nous étudions ici dans quelle mesure la méthode TOPP permet de choisir le chemin offrant le plus de bande passante. Pour cela, nous définissons deux chemins parallèles dans l'émulateur CORE (cf. Figure 6.16). Cette topologie comporte un nœud source s, et deux nœuds destinations t_1 et t_2. On attribue

une capacité identique $C = 10$ Mbps aux deux liens. Nous définissons un trafic de fond UDP entre les nœuds s et t_1, avec une demande $d_1 = 5$ Mbps, de sorte que le lien du haut offre toujours une bande passante de 5Mbps. À cette configuration constante, nous ajoutons sur le lien du bas un trafic de fond UDP de débit d_2 variable.

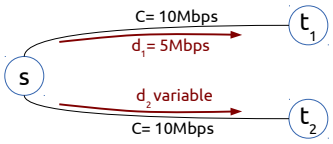


FIGURE 6.16 – Topologie de test sur CORE pour valider les chemins choisis.

Pour chaque valeur de d_2, nous lançons des mesures de bande passante entre s et t_1, et s et t_2, et notons les valeurs estimées pour les deux liens. Nous répétons chaque mesure 100 fois et utilisons une moyenne glissante sur les N dernières mesures pour choisir le chemin offrant la plus grande capacité ($N = 1$ revient à ne pas utiliser de moyenne glissante). Pour chaque valeur de d_2 et de N, nous calculons le pourcentage p de fois où le bon chemin a été choisi.

Un mauvais choix de chemin lorsque la bande passante des deux chemins est proche n'a pas le même impact qu'une erreur pour deux chemins ayant des bandes passantes très différentes. C'est pourquoi nous utilisons la métrique suivante pour évaluer la qualité du choix des chemins obtenu avec TOPP :

$$r = \frac{\text{bande passante obtenue}}{\text{bande passante optimale}} = \begin{cases} \frac{p(C-d_2)+(1-p)(C-d_1)}{C-d_2}, & \text{si } d_2 < d_1 \\ \frac{p(C-d_1)+(1-p)(C-d_2)}{C-d_1}, & \text{si } d_2 > d_1 \end{cases} \tag{6.1}$$

Nous présentons sur la Figure 6.17, la valeur de ce ratio pour l'ensemble de nos tests. On remarque que même sans utiliser de moyenne glissante, la bande passante obtenue est supérieure à 90% de la bande passante optimale. Lorsqu'une moyenne glissante est utilisée, les résultats sont encore améliorés, supérieurs à 95% de la bande passante optimale.

Solution retenue La méthode SQRT n'étant valable que pour le protocole TCP et les estimations obtenues n'étant pas d'une grande qualité, nous nous tournons plutôt vers la méthode de mesure active TOPP. Les résultats obtenus avec l'outil IGI montrent qu'une bonne implémentation de cette méthode permet d'obtenir des estimations fiables.

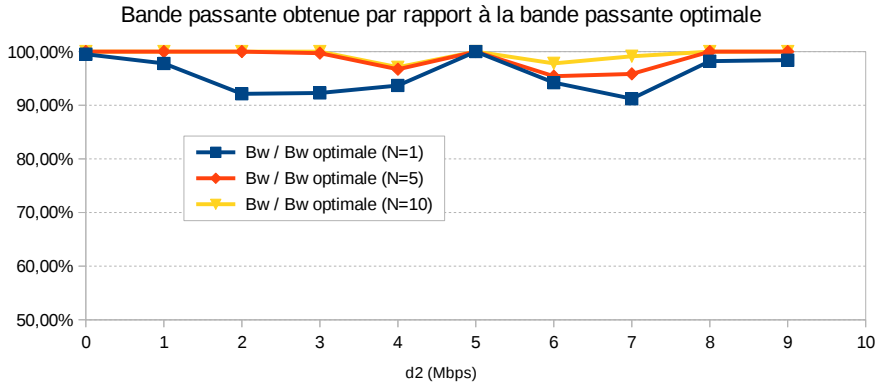


FIGURE 6.17 – Bande passante obtenue par rapport à la bande passante optimale.

6.7 Algorithme d'apprentissage des routes optimales

Nous présentons ci-dessous deux algorithmes qui peuvent être utilisés par l'agent de routage pour piloter l'agent de mesure afin de découvrir les routes à latence minimale [10] dans l'overlay. Le premier permet de mesurer la latence de tous les liens grâce à des paquets sondes envoyés depuis la source. Il n'est toutefois utilisable que pour des réseaux overlays de taille réduite dans la mesure où le nombre de liens à évaluer augmente de façon quadratique avec le nombre de nœuds de l'overlay. Au contraire, le second algorithme utilise une quantité de mesures limitée à chaque pas de temps, mais exploite les observations passées pour apprendre progressivement les chemins optimaux.

6.7.1 Algorithme de couverture du graphe basé sur un cycle eulérien

Comment déterminer la latence de tous les liens de l'overlay en envoyant des paquets sondes depuis la source ? Nous proposons pour cela un algorithme utilisant les dates intermédiaires stockées dans les paquets de mesure. Cet algorithme utilise la notion de cycle eulérien pour parcourir toutes les arêtes du graphe le plus efficacement possible. Un cycle eulérien est un chemin dans un graphe partant d'un sommet quelconque et empruntant exactement une seule fois chaque arête pour revenir au point de départ. Le théorème d'Euler précise qu'il existe un cycle eulérien pour un graphe connexe si et seulement si le degré de chaque nœud du graphe est pair [41]. Pour un graphe complet comme le nôtre, cela signifie que le nombre n de nœuds du graphe doit être impair, ce que nous supposons dans la suite (le cas où n est pair peut être traité en adaptant la technique présentée ci-dessous). L'utilisation d'un algorithme dédié tel que celui proposé par Fleury dans [45] permet de calculer efficacement un cycle eulérien. Si s est le nœud source du cycle, ce cycle eulérien se compose nécessairement de $\frac{n-1}{2}$ sous-cycles repassant

10. Dans cette partie, nous ne considérons que cette métrique.

par s :

$$C = s, a_1, a_2, \ldots, a_{l_1}, s, a_{l_1+1}, \ldots, a_{l_1+l_2}, s, \ldots, s, a_{l_1+l_1+\ldots+l_{\frac{n-1}{2}}} s \tag{6.2}$$

Nous exploitons cette caractéristique pour en extraire les chemins à tester : chacun des $\frac{n-1}{2}$ sous-cycles est redécoupé en chemins de longueur inférieure ou égale à L_{max} (définie dans la section 6.6.1). L'existence de cette longueur maximale de chemin nous oblige nécessairement à répéter quelques arêtes, mais un nombre relativement limité de fois. Nous présentons sur la Figure 6.18 le nombre de chemins nécessaires à la couverture du graphe en fonction de la taille du réseau.

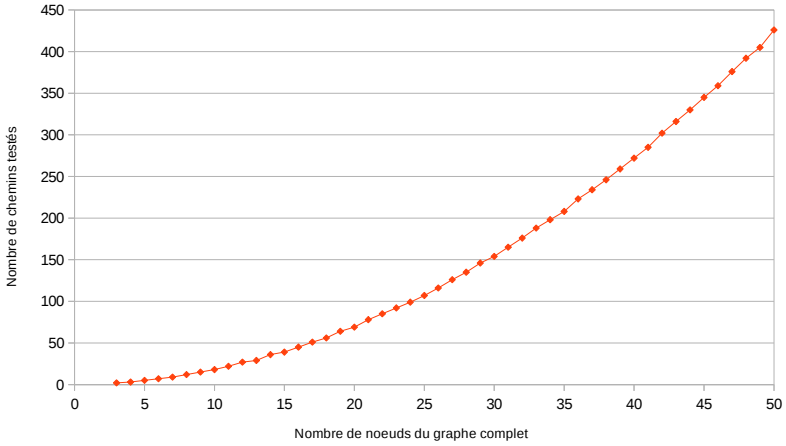


FIGURE 6.18 – Nombre de paquets sondes nécessaires à la couverture du graphe.

Une fois mesurée la latence de chacun des chemins obtenus, nous pouvons construire la matrice d'adjacence représentative des latences sur le réseau, à l'aide des dates tamponnées sur les paquets sondes. Nous pouvons alors utiliser un algorithme de plus court chemin pour déterminer le chemin optimal vers chaque destination. Du fait de la contrainte sur la longueur des chemins, l'algorithme de Dijkstra ne convient pas, et nous utilisons une adaptation de l'algorithme de Bellman Ford décrite dans [87].

Notons toutefois que la perte d'un paquet de mesure sur un chemin entraîne nécessairement la perte des informations pour tous les liens du chemin, le pire cas étant la perte dès le premier lien traversé. Afin de limiter l'influence de ce problème, nous proposons d'insérer une dose d'aléatoire dans la génération du cycle eulérien avec l'algorithme de Fleury, en y modifiant aléatoirement l'ordre de parcours des liens. Cela peut permettre de diversifier légèrement les chemins que nous mesurons.

6.7.2 Algorithme d'apprentissage des routes

L'algorithme basé sur le cycle eulérien mesure la qualité de tous les liens. Le nombre de liens d'un graphe complet de n nœuds étant $n(n-1)/2$, son effort de mesure augmente de façon quadratique avec le nombre de nœuds. En conséquence, une telle approche ne peut pas être utilisée dans un réseau overlay de grande dimension.

On peut se demander s'il est possible découvrir un chemin à latence minimale avec un nombre de mesures bien inférieur, par exemple linéaire ou même logarithmique en n. Ce problème de graphe, connu en anglais sous le nom de *Shortest Path Discovery problem*, a été introduit dans [111] et se pose dans tous contextes où le coût du calcul d'un plus court chemin est faible par rapport à celui de la collecte des métriques des liens. Il peut être formalisé de la façon suivante : étant donné un graphe complet dont les arêtes ont des poids inconnus, mais qui peuvent être déterminés en interrogeant un oracle, comment trouver un plus court chemin entre deux nœuds donnés (et pouvoir garantir que le chemin est optimal) en minimisant le nombre de questions posées à l'oracle ? Dans [25], les auteurs montrent en particulier que pour tout graphe complet de n nœuds, il faut poser au moins $n-1$ questions à l'oracle, et que pour tout algorithme, il existe une mauvaise instance pour laquelle le nombre de questions est supérieur à $n^2/4$.

Ces résultats indiquent qu'il n'est pas envisageable de garantir à chaque pas de mesure qu'un chemin optimal sera découvert si l'on veut préserver la scalabilité du système. Le réseau overlay étant destiné à fonctionner sur de longues durées, on peut par contre exploiter les observations passées pour apprendre progressivement les chemins optimaux avec un effort de mesure raisonnable à chaque pas de temps. Une telle approche peut nous permettre de converger vers les chemins optimaux tout en préservant la scalabilité du système. Comme évoqué au paragraphe 6.3.3, c'est l'approche retenue dans les *Cognitive Packet Networks*, l'algorithme d'apprentissage étant basé sur les réseaux neuronaux et les techniques de Reinforcement Learning [55, 54]. Dans ce paragraphe, nous proposons également un algorithme d'apprentissage, mais basé lui sur une analogie avec les problèmes de type "bandits manchots". Dans ces problèmes, un joueur fait face à un ensemble de machines à sous et doit décider quelles machines jouer, combien de fois il faut jouer chacune et dans quel ordre de manière à maximiser son gain [21, 57]. Dans notre cas, une machine à sous est représentative d'un chemin disponible pour atteindre une destination, et la récompense associée au chemin est égale à l'inverse de sa latence mesurée ($\frac{1}{latence}$).

Les résultats obtenus pour les problèmes de "bandits manchots" stochastiques reposent sur l'hypothèse d'indépendance des récompenses aléatoires associées aux différentes actions. Dans notre cas, cette hypothèse n'est clairement pas satisfaite, certains liens de l'overlay pouvant partager une partie du réseau sous-jacent. Nous optons en conséquence pour la version adversariale du problème dans laquelle aucune hypothèse probabiliste n'est faite sur comment les récompenses associées aux différentes actions

sont générées. En particulier, les récompenses obtenues peuvent dépendre des choix précédents. Auer et al ont proposé dans [17] un algorithme dont le *regret normalisé* (c'est-à-dire la différence entre le gain moyen de l'algorithme sur T étapes et ce qu'on aurait obtenu en jouant toujours la meilleure action) est borné par une quantité proportionnelle à $\sqrt{K \log(K)/T}$, où K est le nombre total d'actions possibles et T représente l'horizon de temps. Cet algorithme, connu sous le nom d'EXP3 (Exploration-Exploitation using Exponential weights), a de plus l'avantage d'être très simple à mettre en œuvre [104]. Comme son nom l'indique, cette stratégie consiste à explorer les actions possibles et à exploiter les observations passées pour pondérer chacune avec un poids exponentiel qui dépend de ses performances relatives dans le passé. Le principe est relativement simple : à chaque action possible est associé un poids à partir duquel est calculé une probabilité de tirage. Un tirage est ensuite effectué conformément à ces probabilités, et l'algorithme stocke la récompense de la machine choisie. Le résultat est exploité en augmentant le poids du chemin choisi grâce à une fonction exponentielle dépendant de la récompense obtenue. Ce fonctionnement est réitéré à chaque nouvel instant.

Cet algorithme est normalement conçu pour ne tester qu'une seule machine à chaque pas de temps. Nous le modifions légèrement pour qu'il soit capable de tester plusieurs chemins à chaque pas de temps.

On note t l'instant présent, $x_i(t)$ la récompense obtenue sur le chemin i à l'instant t, n le nombre de chemins testés en parallèle et K le nombre total de chemins disponibles pour atteindre la destination. La stratégie d'apprentissage suivie est présentée dans l'Algorithme 5, le paramètre γ permettant d'influencer l'exploration des routes. Dans notre implémentation de cet algorithme, nous incluons systématiquement la route IP directe parmi les chemins évalués, les autres étant choisis de manière aléatoire.

Algorithme 5 Algorithme EXP3 modifié.

Require: $\gamma \in (0, 1]$

1: $\omega(i) = 1$, $i = 1 \ldots, K$
2: **for** $t = 1, 2 \ldots T$ **do**
3: Calculer la probabilité de chaque chemin $p_i(t) = (1 - \gamma)\frac{\omega_i(t)}{\sum_{i=1}^{K} \omega_i(t)} + \frac{\gamma}{K}$
4: Tirer n chemins $i_t^1, \ldots, i_t^n$ aléatoirement suivant la distribution de probabilités $p_1(t), \ldots, p_K(t)$
5: Choisir le meilleur chemin $i_t = \text{argmax}\{x_{i_t^1}(t), \ldots, x_{i_t^n}(t)\}$
6: Mettre à jour le poids du chemin $\omega_{i_t}(t+1) = \omega_{i_t}(t) \exp\left(\frac{\gamma x_{i_t}}{K \times p_j(t)}\right)$
7: **end for**

Pour conclure, nous notons qu'il existe des algorithmes offrant de meilleures garanties de performance en termes de regret que la méthode EXP3, et notamment l'algorithme INF proposé dans [16]. Nous notons également que des algorithmes de bandits ont été spécifiquement conçus pour les problèmes de plus courts chemins [59] (voir également le

chapitre 5 de [28]). Par rapport à EXP3, ces algorithmes exploitent le fait que quand les latences des liens du chemin choisi sont révélées, cela fournit également de l'information sur les latences des chemins partageant des liens avec le chemin choisi.

6.8 Plateformes de test et de validation

Nous avons choisi de considérer deux approches complémentaires pour tester et développer une implémentation de la solution proposée : plateforme de virtualisation réseau et plateforme réelle. Nous les présentons dans les deux sections suivantes.

6.8.1 Plateforme virtualisée : émulation réseau

L'utilisation d'une plateforme de virtualisation de réseau permet de créer des réseaux de tests de façon grandement simplifiée par rapport à une plateforme réelle. Elle permet de maîtriser de bout en bout la constitution du réseau et des événements qui s'y déroulent. Deux grands types d'approches peuvent utilisés pour tester et valider le fonctionnement d'un outil réseau sur un réseau virtualisé : la simulation et l'émulation.

La simulation est basée sur une modélisation abstraite et simplifiée, tandis que l'émulation reproduit le fonctionnement complet du réseau, incluant l'ensemble des couches de protocoles traversées (cf section 2.2). Les outils de simulation sont aujourd'hui très utilisés, comme le prouve le grand succès du simulateur d'événement Network Simulator(NS) [63]. Toutefois, l'intégration d'une solution logicielle existante ou en cours de développement dans un simulateur peut se révéler complexe et difficile : il faut développer un modèle de la solution proposée pour l'intégrer au simulateur, et non la réelle implémentation.

C'est pourquoi nous privilégions ici une approche de type émulation : elle permet d'exécuter les programmes existants, comme s'ils fonctionnaient sur une machine et un réseau réel. Un émulateur peut être utilisé comme plateforme pour le développement et le test de l'implémentation en condition quasi-réelle. L'émulation est un processus lourd en calcul, en particulier si l'intégralité d'une machine doit être émulée. Les outils classiques de virtualisation (VMWare, VirtualBox, Xen, KVM) sont relativement lourds, et il reste difficilement envisageable de faire fonctionner plus de quelques machines sur un simple ordinateur de bureau. L'émulation du comportement de deux ou trois machines n'étant pas suffisante pour représenter un vrai réseau, il nous faut considérer un émulateur léger pouvant faire fonctionner plusieurs dizaines de machines simultanément sans trop d'impact sur les performances, et permettant de créer un réseau entre ces machines.

L'émulateur Common Open Research Emulator (CORE) [9] permet d'atteindre assez simplement un tel résultat. Il s'agit d'un outil complet dédié à la création de réseaux virtuels. Il permet de créer des représentations fidèles de réseaux physiques (avec routeurs, switch, pc, etc. . .), avec toutes sortes de protocoles de routage (OSPF, IS-IS, RIP, BGP,

etc. . .) et de services (ssh, apache, etc. . .). Son utilisation est particulièrement simple : une topologie complète peut-être créée et exécutée en l'espace de quelques clics. Dès qu'une topologie est exécutée, le réseau virtuel fonctionne en temps-réel, et il est possible d'exécuter n'importe quel programme sur les machines émulées, sans avoir à leur apporter la moindre modification. Un exemple de topologie créée sur CORE pour nos tests est présenté sur la Figure 6.19.

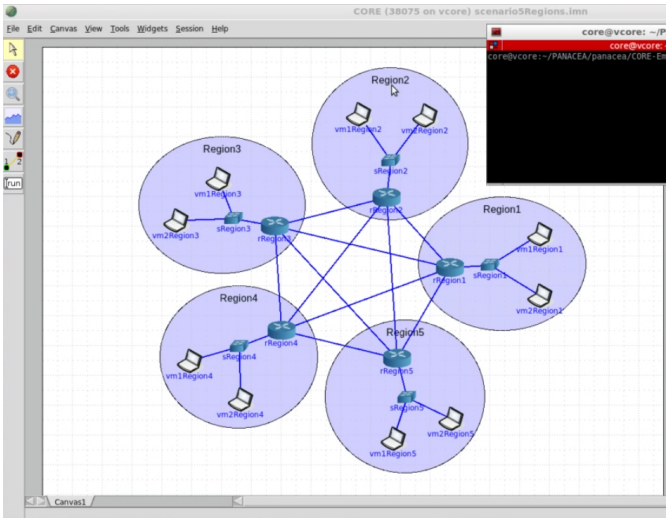


FIGURE 6.19 – Interface principale de l'émulateur CORE.

La légèreté du processus d'émulation obtenu avec cet émulateur repose sur la technologie LXC (Linux Containers) [7] qui fournit une isolation légère des processus de chaque machine émulée. Chaque machine fonctionne dans un conteneur séparé, qui apporte une virtualisation plus ou moins complète de l'environnement exécution (processeur, mémoire vive, réseau), mais pas de la machine en elle-même. Les programmes exécutés sur les machines virtualisées partagent directement les ressources de la machine hôte, sans surcoût notable dû à l'isolation. L'émulateur CORE gère de plus la création du réseau entre les machines virtualisées, en autorisant les modifications en temps-réel des caractéristiques des liens qui le compose (délai, pertes, capacité, duplications des paquets, . . .). Malgré tout, l'exécution de fonctions proches du système peuvent nécessiter une modification du noyau du système d'exploitation de l'hôte pour fonctionner correctement sur les machines virtualisées. Les développeurs de CORE fournissent des patchs pour résoudre ce type de difficulté. Nous avons dû utiliser l'un de ces patchs pour faire fonctionner correctement l'interception des paquets sur les machines virtualisées.

6.8.2 Plateforme réelle : Amazon EC2

Amazon Elastic Compute Cloud (EC2)[6] est un service payant permettant à des tiers de louer des serveurs sur lesquels ils peuvent exécuter leurs propres applications. EC2 permet un déploiement facile des applications via une interface web et une API par lesquelles un client peut créer, lancer, et arrêter des instances de serveurs en fonction de ses besoins. L'utilisateur peut choisir parmi plusieurs configurations matérielles et plusieurs systèmes d'exploitation. L'utilisateur paye en fonction du temps d'utilisation des serveurs. Les serveurs du service EC2 sont situés dans 8 centres de données répartis dans le monde (cf. Figure 6.20). Grâce à ces différents sites, un client peut mettre en place des instances de serveurs éloignés physiquement les unes des autres et interconnectées par notre réseau overlay.

FIGURE 6.20 – Les différents sites Amazon disponibles.

6.9 Validation de l'implémentation

Dans cette partie, nous nous attardons sur la validation fonctionnelle de notre implémentation. Nous nous concentrons sur la principale métrique aujourd'hui intégrée à notre implémentation, à savoir la mesure de latence. Nous commençons tout d'abord par introduire l'application client utilisée pour nos observations.

6.9.1 Application de ping utilisant UDP

Pour la validation du fonctionnement d'optimisation du délai des chemins considérés, il est nécessaire d'utiliser une application mesurant les délais du point de vue du

client. Nous avons délibérément décider de ne pas utiliser l'utilitaire ping, car ce dernier repose sur l'utilisation du protocole ICMP, dont la priorité peut-être altérée sur certains équipements réseaux par rapport à d'autres types de trafic.

Pour observer les gains obtenus en activant le réseau overlay, nous souhaitons observer les délais obtenus dans deux configurations : chemin Internet classique et chemin dans le réseau overlay. Comme l'encapsulation des paquets sur le réseau overlay repose sur le protocole UDP, et pour ne pas biaiser les comparaisons, nous choisissons d'utiliser une application cliente reposant elle aussi sur le protocole UDP. Ceci nous a naturellement conduit à développer par nous-mêmes un utilitaire de mesure similaire à la commande ping, mais utilisant le protocole UDP en lieu et place du protocole ICMP. Nous utiliserons cette application pour nos différentes expérimentations.

6.9.2 Estimation du surcoût temporel

Nous présentons ici les résultats de mesure du surcoût temporel (overhead) introduit par l'utilisation du système Panacea par rapport à un routage normal. Pour cela nous mesurons les délais effectifs en utilisant l'application de ping UDP que nous avons développé, en activant ou non le système Panacea. Nous utilisons la plateforme d'émulation avec des topologies linéaires de différentes tailles $N = 2, \ldots, 5$, comme présenté sur la Figure 6.21.

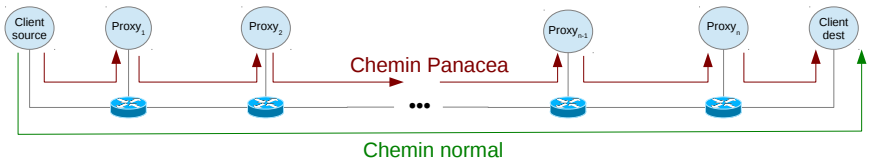


FIGURE 6.21 – Topologies testées pour mesurer l'overhead.

On fixe un délai nul sur l'ensemble des liens de la topologie. Pour chacune des tailles, nous mesurons le délai avec et sans l'interception Panacea. Lorsque l'interception est fonctionnelle, nous forçons le système Panacea à router le paquet en passant par l'ensemble des proxies disponibles avant d'atteindre la destination. Avec ces mesures, nous avons pu observer une latence supplémentaire d'environ 3 ms de bout-en-bout par rapport au routage normal, et ce quel que soit la taille de la topologie. Il semble que la plus grande partie de ce temps soit passée dans la phase d'interception et d'encapsulation.

Remarque 6.9.1. *Pour estimer correctement l'impact global du système, il faut aussi tenir compte de la latence constatée entre la machine cliente et le proxy correspondant. Les deux algorithmes que nous proposons tiennent compte du surcoût total introduit par le système (overhead évalué ci-dessus + latence client/proxy) pour déterminer la meilleure route.*

6.9.3 Optimisation de la latence

Cette section est consacrée à la validation du fonctionnement de l'architecture et de l'implémentation proposée pour l'optimisation de la latence, qui constitue la seule métrique complètement implémentée aujourd'hui dans le système. Nous commençons par valider le fonctionnement sur la plateforme d'émulation avec l'outil CORE, et nous terminons par une expérimentation de 4.5 jours sur la plateforme Amazon. Les résultats obtenus montrent que l'architecture et l'implémentation proposée sont fonctionnelles et permettent effectivement de réduire la latence ressentie par l'utilisateur final. Pour observer le comportement sur un plus grand réseau, nous terminons cette partie par une observation des résultats pouvant être obtenus avec l'algorithme EXP3, via une simulation hors-ligne du système en utilisant les informations de latence du réseau NLNOG.

6.9.3.1 Validation sur l'émulateur CORE

Nous considérons ici une topologie comportant 5 sites distincts, comportant chacun un proxy et une machine cliente exécutant les agents de notre système. Nous cherchons à mesurer le RTT entre la machine cliente de la région 1 et la machine cliente de la région 2. Nous utilisons notre application de ping UDP pour mesurer ce RTT en installant un serveur sur la machine 2 et un client sur la machine 1. En parallèle, nous exécutons également un deuxième couple client/serveur de l'application ping UDP, mais fonctionnant sur un port UDP non intercepté par les agents de transmission. On peut ainsi comparer directement les performances obtenues avec et sans le routage overlay. Des mesures sont lancées automatiquement toutes les minutes sur les deux couples client/serveur. Les mesures de latence effectuées par les proxies sont aussi menées chaque minute.

Le scénario expérimental est décrit sur la Figure 6.22. Dans la topologie de départ (étape 1), tous les liens ont une latence de 25ms et le chemin direct est donc le plus court pour atteindre le nœud 2 depuis le nœud 1. Cinq minutes plus tard (étape 2), la latence du lien 1-2 passe à 125ms, et le routage overlay se fait avec deux sauts. Les étapes 3 et 4 ont lieu 5 minutes et 10 minutes plus tard, respectivement, et force le système à utiliser un chemin de longueur 3 puis 4.

Les mesures de RTT obtenues sont présentées sur la Figure 6.23. On peut observer les 4 paliers correspondant aux 4 étapes de l'expérience. À partir du premier événement d'augmentation du délai, on constate que notre système arrive à rediriger le trafic sur un chemin ayant une latence inférieure à la route IP. Sur ce scénario, l'algorithme d'apprentissage EXP3 n'arrive pas toujours à découvrir le chemin optimal du fait du nombre limités de mesures entre chaque étape. On constate néanmoins que cet algorithme permet de réduire significativement la latence par rapport à la route IP directe.

Cette expérimentation valide le fonctionnement global du système, de l'interception des paquets jusqu'au routage de ces derniers, mais le scénario reste limité. C'est pourquoi nous allons maintenant présenter des tests en situations réelles effectués sur la plateforme

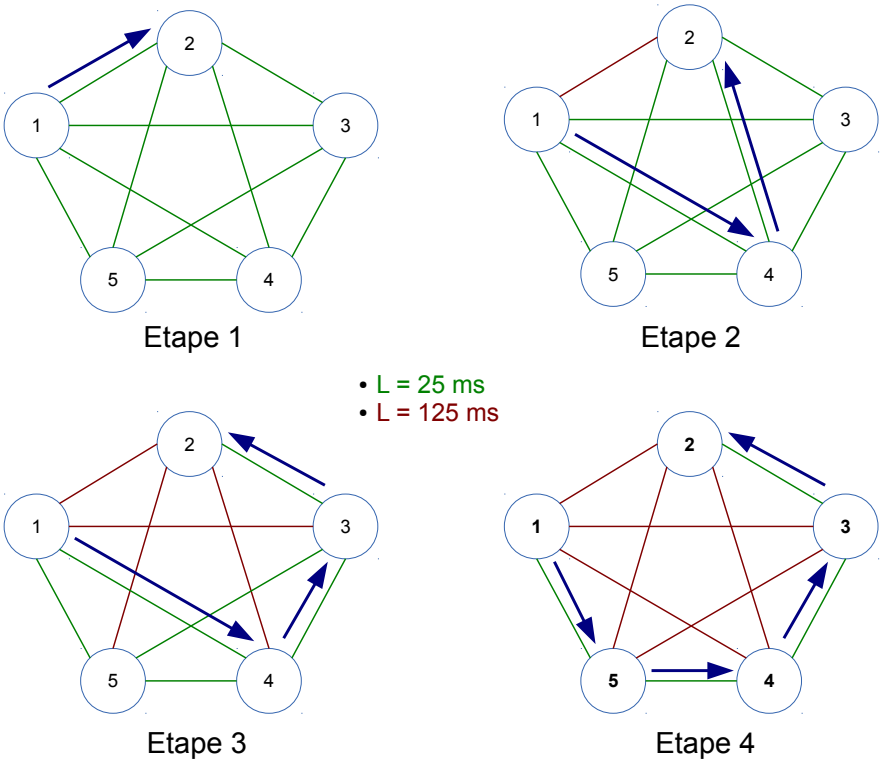


FIGURE 6.22 – Scénario de test sur l'émulateur CORE.

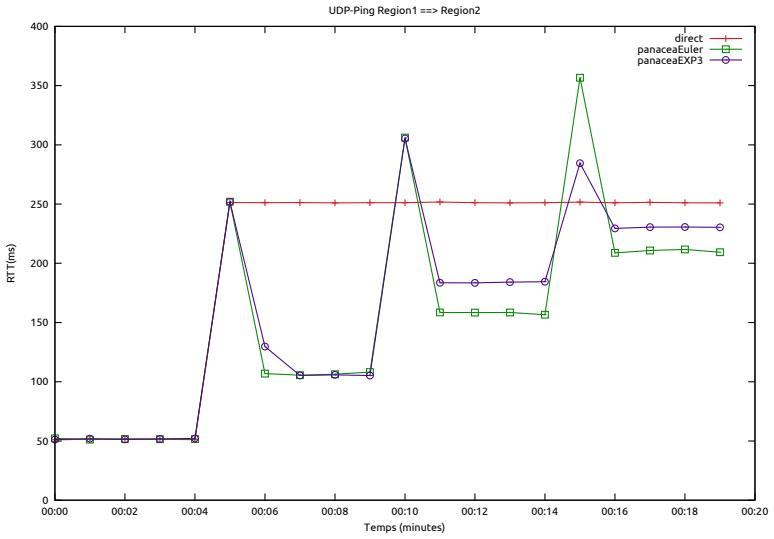


FIGURE 6.23 – Validation du fonctionnement avec les latences sur l'émulateur CORE.

Amazon.

6.9.3.2 Validation sur Amazon

Nous présentons ci-dessous les résultats expérimentaux obtenus avec les 8 sites de la plateforme Amazon. Comme avec l'émulateur CORE, nous utilisons deux couples de client/serveur pour l'application de ping UDP. Dans cette expérience, nous mesurons par contre les délais pour chacun des 56 couples origine/destination afin d'observer le comportement sur l'ensemble du réseau overlay. Bien entendu ici, nous ne maîtrisons pas les événements pouvant subvenir sur le réseau. Les RTT sont mesurés toutes les deux minutes, sur une durée de 4.5 jours. Sur cette plateforme, nous n'avons pour l'instant testé que l'algorithme exhaustif (Euler). Pour cette taille de topologie, le nombre de liens à mesurer reste relativement raisonnable (56 liens).

Nous présentons sur la Figure 6.24 une courbe représentative du type de résultat pouvant être obtenus lorsque le lien internet propose une latence particulièrement sous-optimale. On y observe que le réseau overlay a permis de diminuer significativement le délai constaté entre les villes de SaoPaulo et Tokyo [11]. À l'inverse lorsqu'il n'est pas possible d'améliorer la latence, nous avons bien pu vérifier que notre système n'introduit aucun surcoût grâce à l'interception adaptative des paquets.

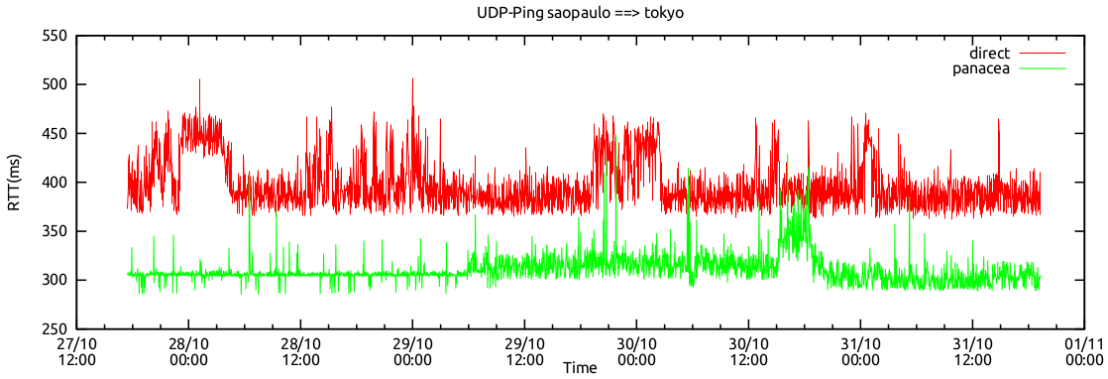


FIGURE 6.24 – Amélioration des délais entre régions éloignées.

6.9.3.3 Observation du comportement de l'algorithme EXP3

Nous observons ici le comportement du système pour une taille de topologie plus importante. Pour cela, nous mettons à profit les traces obtenues sur le réseau NLNOG,

11. Nous avons remarqué que les latences des routes passant dans l'océan pacifique pouvaient être significativement réduites avec notre système.

composé de 20 nœuds. Le nombre de liens à mesurer devient ici plus important (190 liens), et commence à justifier l'utilisation d'un algorithme minimisant l'effort de mesure.

Nous procédons à des simulations hors-ligne reproduisant le fonctionnement du système et des différents algorithmes en utilisant les données de latence du réseau NLNOG. Les simulations tiennent compte du délai supplémentaire introduit par l'overlay, que nous fixons ici de manière pessimiste à 5 ms. Nous cherchons à étudier les performances de l'algorithme EXP3 par rapport au routage internet classique et à l'algorithme exhaustif (Euler). Les résultats moyens obtenus avec l'algorithme EXP3 sont résumés dans la Table 6.2. Nous avons considéré des tirages de 2, 5 et 10 chemins par destination. Comme un chemin est composé de $L_{max} = 4$ liens au maximum, nous savons que l'algorithme EXP3 ne mesurera au maximum que 8, 20 ou 40 liens pour ces 3 configurations (à comparer aux 190 liens du réseau).

TABLE 6.2 – Comportement moyen sur toute la topologie NLNOG de l'algorithme EXP3 par rapport au routage optimal (Euler).

	Direct	**2 tirages**	**5 tirages**	**10 tirages**
Instants non optimaux(%)	22.27	22.11	18.45	15.2
Écart à l'optimal(%)	9.7	8.98	7.97	6.62

On observe tout d'abord que pour 22.27% des cas [12], le chemin direct est moins intéressant que le chemin optimal fourni par l'algorithme du cycle eulérien. On observe ensuite l'influence du nombre de tirages effectué par l'algorithme EXP3 : on confirme bien qu'en augmentant le nombre de tirages, on se rapproche du routage optimal.

Ces valeurs moyennes ne permettent pas de mesurer véritablement le gain obtenu dans les situations de routage pathologiques que nous cherchons à améliorer. Nous présentons dans la Table 6.3 quelques exemples pour lesquels le système permet d'obtenir un gain intéressant, malgré la couverture partielle du graphe.

Ces exemples mettent mieux en évidence l'influence du nombre de tirages sur les résultats obtenus. Pour cette topologie de $N = 20$ nœuds, on constate une nette amélioration par rapport au routage Internet à partir de 5 tirages ($N/4$ pour cette topologie).

La Figure 6.25 présente l'exemple du lien Japon-Chili lorsque l'on utilise 5 tirages pour l'algorithme EXP3. On constate que l'algorithme converge très progressivement vers la route optimale, et ce même si des événements réseaux viennent perturber les résultats au milieu de la semaine observée.

12. Cette valeur est plus faible que les 38% évoqués dans l'introduction, car les algorithmes de routage implémentés tiennent compte du surcoût total introduit par le système (10 ms pour un RTT : $2 \times 3 = 6$ ms pour l'overhead + une valeur arbitraire de 4×1 ms pour les latences client/proxy) pour choisir la meilleure route

TABLE 6.3 – Écart (%) du RTT obtenu par rapport à l'optimal (Euler) sur quelques cas où le routage internet est sous-optimal.

	Direct	**2 tirages**	**5 tirages**	**10 tirages**
Singapour-Israël	19.58	15.84	11.08	4.25
Japon-Chili	48.96	39.20	8.97	14.66
Australie-Chili	20.86	13.73	7.28	4.18
Norvège-Singapour	17.41	9.85	5.56	1.89
Pologne-Brésil	12.05	11.67	6.63	2.90
Irlande-Moscou	77.78	59.30	32.49	16.05
Israël-Moscou	37.69	29.60	13.76	5.27

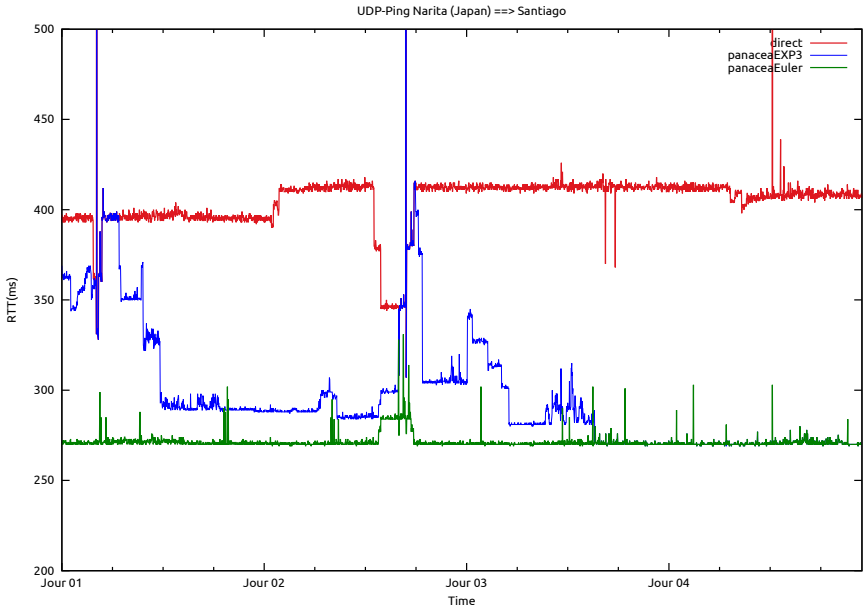


FIGURE 6.25 – NLNOG lien Japon-Chili, 5 tirages pour l'algorithme EXP3.

6.10 Conclusion et perspectives

Nous avons présenté dans ce chapitre une solution novatrice de réseau overlay fonctionnant au niveau applicatif et permettant d'améliorer le routage internet entre différents clouds. Cette solution a l'avantage de ne nécessiter aucun changement, ni de l'application cliente ni au niveau du système d'exploitation. Nous avons pu valider l'implémentation à la fois sur une plateforme émulée et sur une plateforme réelle déployée dans l'Internet. Cette dernière nous a permis de mettre en évidence les gains significatifs qui peuvent être obtenus pour certains liens Internet pour lesquels le routage IP est

particulièrement sous-optimal. Nous avons proposé et implémenté deux algorithmes de pilotage des mesures utilisant des philosophies bien distinctes. L'algorithme basé sur le cycle eulérien garantit l'optimalité des chemins obtenus mais requiert un nombre de mesures quadratique en la taille du réseau overlay. Les résultats obtenus avec l'algorithme EXP3 indiquent l'intérêt d'une stratégie d'apprentissage permettant de découvrir les chemins optimaux dans des réseaux overlays de grande dimension.

Un certain nombre de travaux devront être menés à bien pour compléter la solution existante et sa validation. Il nous paraît essentiel de concevoir des algorithmes d'apprentissage plus adaptés au problème qui exploitent les latences mesurées sur un chemin pour en déduire de l'information sur la qualité des chemins ayant des liens communs avec le chemin mesuré. Sur un plan plus pratique, l'implémentation d'une méthode de mesure de la bande passante est également un point important. Nous envisageons aussi de faire évoluer l'architecture du système pour la rendre plus modulaire, en séparant physiquement l'agent de mesure de l'agent de routage, pour autoriser la création de plusieurs agents de routage distincts sur un même cloud. À terme, la solution développée sera mise à disposition de la communauté sous licence open source.

7
Conclusion

Les réseaux de communication constituant Internet doivent faire face à l'afflux grandissant de nouveaux utilisateurs. Les services proposés continuent à se développer et nécessitent de plus en plus de bande passante et de qualité de services. Les capacités de transport de données offertes par les opérateurs étant limitées, ces derniers doivent procéder à une optimisation de leurs ressources existantes pour faire face à la demande. La volatilité des trafics limite de plus en plus la pertinence d'une optimisation hors-ligne de la configuration réseau, et justifie la proposition de méthodes d'optimisation dynamique du routage.

Nous avons proposé dans cette thèse trois méthodes d'optimisation dynamique du routage permettant de répondre aux variations temporelles du trafic, pour trois cas d'utilisation distincts :

— optimisation du routage intra-domaine OSPF en modifiant les poids des liens
— optimisation du placement des LSP dans les réseaux utilisant la technologie MPLS
— optimisation du routage inter-domaine entre clouds via la mise en place d'un réseau overlay logiciel

Les résultats d'expérimentation montrent que la méthode proposée pour l'optimisation dynamique du routage dans les réseaux OSPF, associée à une estimation robuste des trafics permet de diminuer significativement la congestion réseau dans les réseaux OSPF par rapport à une configuration statique. Quelques pistes peuvent être envisagées pour améliorer encore la qualité des résultats obtenus. Tout d'abord, les contraintes temporelles nous ont conduit à choisir la méthode tomogravité, fournissant rapidement une

estimation des trafics, mais potentiellement assez peu précise. Il pourrait être intéressant d'étudier d'autres méthodes plus poussées d'estimation de la matrice de trafic. Dans le même ordre d'idée, l'algorithme en ligne étant censé fonctionner sur de longues durées, il pourrait être bénéfique d'introduire une forme d'apprentissage progressif de l'évolution des trafics au cours du temps pour rendre le système encore plus réactif.

Les travaux menées sur l'optimisation des réseaux MPLS nous ont amené à proposer un algorithme inspiré de la théorie des jeux permettant d'optimiser efficacement le placement des LSP. L'algorithme proposé permet d'obtenir des solutions de placement de bonne qualité, avec un effort de calcul bien plus restreint que pour les autres méthodes testées. Certains résultats théoriques ont de plus pu être établis concernant cet algorithme. Dans la continuité de ces travaux, il serait intéressant d'étudier les garanties de performances de l'algorithme proposé pour des fonctions coût autre que linéaire ou quadratique. Il parait aussi très important d'introduire une forme d'incertitude sur les demandes, comme nous avons pu le faire pour les réseaux OSPF, pour prendre en compte l'évolution future des trafics.

Enfin, l'architecture et l'implémentation proposée pour la mise en place d'un réseau overlay auto-guérissant et auto-optimisant entre plateformes de clouds permet d'optimiser le routage internet lorsque ce dernier est sous-optimal ou défaillant. Les résultats que nous avons obtenus nous ont permis de valider notre architecture et montrent que des gains notables peuvent être obtenus par rapport au routage internet classique pour l'optimisation de la latence. Nos simulations sur un algorithme d'apprentissage des routes montrent qu'il est possible de limiter l'effort de mesure, tout en continuant à améliorer la latence par rapport à un chemin internet sous-optimal. Ces résultats sont particulièrement prometteurs et un certain nombre de travaux restent encore à mener pour parfaire la solution proposée. Il sera tout d'abord nécessaire de terminer l'implémentation des mesures de bande passante de pertes. L'utilisation d'algorithmes d'apprentissage plus évolués est une piste sérieuse que nous avons commencé à considérer. L'implémentation de méthodes inspirées des réseaux de neurones ou d'améliorations de l'algorithme EXP3 pourrait permettre de réduire significativement le temps de convergence vers les chemins optimums, tout en conservant une bonne scalabilité du système. Nous envisageons aussi de développer des applications utilisant directement le réseau overlay : une application de copie de fichier multichemins minimisant le temps de copie entre deux sites en est un bon exemple. Enfin, à terme, nous souhaitons mettre notre implémentation à disposition de la communauté sous licence open-source.

Glossaire

AS Système Autonome. 7–9, 94

BGP Border Gateway Protocol. 9, 94, 118

CORE Common Open Research Emulator. 105, 109, 110, 112, 118, 119

EGP Exterior Gateway Protocol. 9

FEC Forward Equivalent Class. 11

ICMP Internet Control Message Protocol. 94, 108, 121

IETF Internet Engineering Task Force. 7, 9

IGP Interior Gateway Protocol. 8, 9, 35

IP Internet Protocol. 5–7, 10–12, 21, 35, 37, 40, 93–95, 97–99, 101, 103, 105–108, 117, 122, 126

IS-IS Intermediate System to Intermediate System. 8, 35, 118

ISO Organisation internationale de normalisation. 6

LER Label Edge Router. 10

LSP Label Switching Path. 11, 65–67, 82, 85, 86, 89, 90

LSR Label Switch Router. 10

MPLS Multi Protocol Label Switching. 10, 11, 65, 67, 81, 93

NAT Network Address Translation. 108

OSI Open Systems Interconnection. 6, 7, 10

OSPF Open Shortest Path First. 8, 9, 35, 37, 53, 118

RIP Routing Information Protocol. 8, 118

RTT Round Trip Time. 14, 109, 110, 122, 124, 125

SLoPS Self-Loading Periodic Streams. 18

SNMP Simple Network Management Protocol. 37, 40, 41, 43, 61, 63, 85

TCP Transmission Control Protocol. 12, 15–18, 108, 109, 113

TOPP Trains of packet pairs. 19, 20, 109, 110

UDP User Datagram Protocol. 11, 12, 16, 18, 106, 108, 110, 111, 113, 121, 122, 124

Bibliographie

[1] Boston University Representative Internet Topology Generator (BRITE). http://www.cs.bu.edu/brite/, 2001.

[2] ILOG CPLEX. http://www.ilog.com/products/cplex/, 2012.

[3] Library for Efficient Modeling and Optimization in Networks (LEMON). https://lemon.cs.elte.hu/trac/lemon, 2012.

[4] Software-defined networking : The new norm for networks. White paper. Open Networking Foundation, April 13 2012.

[5] Open dove. https://wiki.opendaylight.org/view/Open_DOVE:Main, 2013.

[6] Amazon ec2. http://aws.amazon.com/fr/ec2/, 2014.

[7] Lxc containers. https://linuxcontainers.org/lxc/introduction/, 2014.

[8] Netfilter/iptables. http://www.netfilter.org/, 2014.

[9] J. Ahrenholz, C. Danilov, T.R. Henderson, and J.H. Kim. Core : A real-time network emulator. In *Military Communications Conference, 2008. MILCOM 2008. IEEE*, pages 1–7, Nov 2008.

[10] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows*. Prentice Hall, 1993.

[11] A Al Sheikh, O. Brun, P.E. Hladik, and B.J. Prabhu. A best-response algorithm for multiprocessor periodic scheduling. In *Real-Time Systems (ECRTS), 2011 23rd Euromicro Conference on*, pages 228–237, July 2011.

[12] A. Altin, P. Belotti, and M.Ç. Pinar. Ospf routing with optimal oblivious performance ratio under polyhedral demand uncertainty. *Optimization and Engineering*, 2009.

[13] A. Altin, B. Fortz, and H. Ümit. Oblivious ospf routing with weight optimization under polyhedral demand uncertainty. In *International Network Optimization Conference (INOC 2009)*, Pisa, Italy, April 26-29 2009.

[14] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. In *Proceedings of the Eighteenth ACM Symposium on Operating Systems Principles*, SOSP '01, pages 131–145, New York, NY, USA, 2001. ACM.

[15] K. Andreev, B. M. Maggs, A. Meyerson, and R. Sitaraman. Designing overlay multicast networks for streaming. In *Proceedings of the Fifteenth Annual ACM Symposium on Parallel Algorithms and Architectures (SPAA), San Diego, CA, USA*, June 2003.

[16] J-Y Audibert and S. Bubeck. Regrets bounds and minimax policies under partial monitoring. *Journal of Machine Learning Research*, 11 :2785–2836, 2010.

[17] P. Auer, N. Cesa-Bianchi, Y. Freund, and R. Schapire. The non-stochastic multi-armed bandit problem. *SIAM Journal on Computing*, 32(1) :48–77, 2002.

[18] Baruch Awerbuch and et al. The price of routing unsplittable flow, 2005.

[19] S. Banerjee, B. Bhattacharjee, C. Kommareddy, and G. Varghese. Scalable application layer multicast. In *Proc. of the ACM SIGCOMM*, New York, USA, 2002.

[20] M. Beck, T. Moore, and J.S. Plank. An end-to-end approach to globally scalable programmable networking. In ACM Press, editor, *in Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, 2003.

[21] D.A. Berry and B. Fristedt. *Bandit problems : Sequential allocation of experiments.* Monographs on Statistics and Applied Probability. Chapman & Hal, London, 1985.

[22] Erik Gunnar Boman. *Infeasibility and Negative Curvature in Optimization.* PhD thesis, Stanford, CA, USA, 1999. AAI9924533.

[23] R. Braden. *RFC 1122 Requirements for Internet Hosts - Communication Layers.* Internet Engineering Task Force, October 1989.

[24] O. Brun and J. M. Garcia. Dynamic igp weight optimization in ip networks. In *First International Symposium on Network Cloud Computing and Applications (NCCA)*, 2011.

[25] Olivier Brun, Josu Doncel, and Christopher Thraves Caro. Shortest path discovery problem, revisited (query ratio, upper and lower bounds). Research report, LAAS-CNRS, October 2014.

[26] J. Cao, D. Davis, S. Wiel, and B. Yu. Time-varying network tomography : Router link data. Technical report, Bell Labs, February 2000.

[27] E. Cela. *The Quadratic Assignment Problem : Theory and Algorithms.* Kluwer Academic Publishers, 1998.

[28] N. Cesa-Bianchi and G. Lugosi. *Prediction, Learning and Games*. Cambridge University Press, 2006.

[29] J. Chu and C. T. Lea. Optimal link weights for ip-based networks supporting hose-model vpns. *IEEE/ACM Transactions on Networking*, 17(3) :778–786, June 2009.

[30] Y.H. Chu, S.G. Rao, and H. Zhang. A case for end system multicast. In ACM, editor, *ACM SIGMETRICS 2000*, pages 1–12, Santa Clara, CA, June 2000.

[31] Cisco Systems. Sampled NetFlow. http://www.cisco.com/en/US/docs/ios/12_0s/feature/guide/12s_sanf.html, 2001.

[32] Cisco Systems. Cisco ios netflow. http://www.cisco.com/en/US/products/ps6601/products_ios_protocol_group_home.html, 2004.

[33] Andy Collins. The detour framework for packet rerouting. Technical report, 1998.

[34] Pierre Coucheney. *Auto-optimisation des réseaux sans fil : une approche par la théorie des jeux*. PhD thesis, Université de Grenoble, 2006.

[35] Mike Dahlin, Bharat Chandra, Let Gao, and Amol Nayate. End-to-end wan service availability. In *In Proc. 3rd USITS*, pages 97–108, 2001.

[36] J.L. Deneubourg, J.M. Pasteels, and J.C. Verhaeghe. Probabilistic behaviour in ants : A strategy of errors ? *Journal of Theoretical Biology*, 105(2) :259 – 271, 1983.

[37] E.W. Dijkstra. A note on two problems in connexion with graphs. *Numerische Mathematik*, 1(1) :269–271, 1959.

[38] Marco Dorigo. Optimization, learning and natural algorithms. *Ph. D. Thesis, Politecnico di Milano, Italy*, 1992.

[39] Marco Dorigo, Vittorio Maniezzo, and Alberto Colorni. The ant system : Optimization by a colony of cooperating agents. *26th IEEE Transactions on Systems, Man, and Cybernetics - part B*, (1) :29–41, 1996.

[40] RW Eglese. Simulated annealing : a tool for operational research. *European journal of operational research*, 46(3) :271–281, 1990.

[41] L. Euler. Solutio problematis ad geometriam situs pertinentis. *Commentarii academiae scientiarum Petropolitanae 8*, pages 128–140, 1741.

[42] N. Feamster, H. Balakrishnan, J. Rexford, A. Shaikh, and J. van der Merwe. The case for separating routing from routers. In ACM Press, editor, *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, 2004.

[43] A. Feldmann, O. Maennel, Z. Morley Mao, A. Berger, and B. Maggs. Locating internet routing instabilities. In *Proceedings of the ACM SIGCOMM 2004 Conference (SIGCOMM), Portland, Oregon, USA*, August 2004.

[44] R. Fletcher and C. M. Reeves. Function minimization by conjugate gradients. *The Computer Journal*, 7(2) :149–154, 1964.

[45] Fleury. Deux problemes de geometrie de situation. *Journal de mathematiques elementaires*, pages 257–261, 1883.

[46] C. Fortuny, O. Brun, and J. M. Garcia. Metric optimization in IP networks. In *19th International Teletraffic Congress*, pages 1225–1234, Beijing, China, 2005.

[47] C. Fortuny, O. Brun, and J. M. Garcia. Fanout inference from link counts. In *4th European Conference on Universal Multiservice Networks (ECUMN 2007)*, pages 190–199, 2007.

[48] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *Proc. 19th IEEE Conf. on Computer Communications (INFOCOM)*, 2000.

[49] B. Fortz and M. Thorup. Increasing internet capacity using local search. *Computational Optimization and Applications*, 29 :13–48, 2004.

[50] B. Fortz and H. Ümit. Efficient techniques and tools for intra-domain traffic engineering. Technical Report 583, ULB Computer Science Departement, 2007.

[51] Drew Fudenberg and Jean Tirole. *Game Theory*. MIT Press, Cambridge, MA, 1991.

[52] J.M. Garcia, A. Rachdi, and O. Brun. Optimal LSP placement with QoS constraints in DiffServ/MPLS networks. In *Proceedings of the 18th International Teletraffic Congress - ITC-18*, volume 5 of *Teletraffic Science and Engineering*, pages 11 – 20. 2003.

[53] E. Gelenbe. Cognitive packet networks. US Patent 6804201 B1, 2004.

[54] E. Gelenbe and Z. Kazhmaganbetova. Cognitive packet network for bilateral asymmetric connections. *IEEE Trans. Industrial Informatics*, 10(3) :1717–1725, 2014.

[55] E. Gelenbe, R. Lent, A. Montuori, and Z. Xu. Towards networks with cognitive packets. In *Proc. 8th Int. Symp. Modeling, Analysis and Simulation of Computer*

and Telecommunication Systems (IEEE MASCOTS), San Francisco, CA, USA, pages pp 3–12, August 29-September 1 2000.

[56] M. Gellman. *QoS Routing for Real-time Traffic*. PhD thesis, Imperial College London, 2007.

[57] J.C. Gittins. *Multi-armed bandit allocation indices*. Wiley-Interscience Series in Systems and Optimization. John Wiley & Sons, Chichester, 1989.

[58] J. Gross. Programmable networking with open vswitch. LinuxCon - http ://events.linuxfoundation.org/sites/events/files/slides/OVS-LinuxCon%202013.pdf, September 2013.

[59] A. Gyorgy, T. Linder, G. Lugosi, and G. Ottucsak. The on-line shortest path problem under partial monitoring. *Journal of Machine Learning Research*, 8 :2369–2403, 2007.

[60] J. Han and F. Jahanian. Impact of path diversity on multi-homed and overlay networks. In *In Proceedings of IEEE International Conference on Dependable Systems and Networks*, 2004.

[61] Ningning Hu, Student Member, Peter Steenkiste, and Senior Member. Evaluation and characterization of available bandwidth probing techniques. *IEEE Journal on Selected Areas in Communications*, 21 :879–894, 2003.

[62] I.Juva, S.Vaton, and J.Virtamo. Quick traffic matrix estimation based on link count covariances. *2006 IEEE International Conference on Communications (ICC 2006),Istanbul*, 2006.

[63] Teerawat Issariyakul and Ekram Hossain. *Introduction to Network Simulator NS2*. Springer Publishing Company, Incorporated, 1 edition, 2008.

[64] Manish Jain and Constantinos Dovrolis. End-to-end available bandwidth : Measurement methodology, dynamics, and relation with TCP throughput. In *In Proceedings of ACM SIGCOMM*, pages pp. 295–308, 2002.

[65] Leonard Kleinrock. *Theory, Volume 1, Queueing Systems*. Wiley-Interscience, 1975.

[66] Joseph B. Kruskal. On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem. *Proceedings of the American Mathematical Society*, 7(1) :48–50, February 1956.

[67] C. Labovitz, R. Malan, and F. Jahanian. Internet routing instability. *IEEE/ACM Transactions on Networking*, 6(5) :515–526, 1998.

[68] Craig Labovitz, Abha Ahuja, Abhijit Bose, and Farnam Jahanian. Delayed internet routing convergence. *SIGCOMM Comput. Commun. Rev.*, 30(4) :175–187, August 2000.

[69] T. Leighton. Improving performance on the internet. *Communications of the ACM*, 52(2), February 2009.

[70] J. Liebeherr and T. K. Beam. Hypercast : A protocol for maintaining multicast group members in a logical hypercube topology. In *Proceedings of the First International COST264 Workshop on Networked Group Communication*, pages 72–89. Springer-Verlag, 1999.

[71] G. Malkin. RFC 2453 : RIP Version 2 . Technical report, IETF, 1998.

[72] Rafael Martí. Multi-start methods. In *Handbook of metaheuristics*, pages 355–368. Springer, 2003.

[73] Matthew Mathis, Jeffrey Semke, Jamshid Mahdavi, and Teunis Ott. The macroscopic behavior of the tcp congestion avoidance algorithm. *SIGCOMM Comput. Commun. Rev.*, 27(3) :67–82, July 1997.

[74] A. Medina, N. Taft, S. Battacharya, C. Diot, and K. Salamatian. Traffic matrix estimation : Existing techniques compared and new directions. In *SIGCOMM, Pittsburgh*, August 2002.

[75] Bob Mel, Mats Björkman, and Per Gunningberg. Regression-based available bandwidth measurements. In *In International Symposium on Performance Evaluation of Computer and Telecommunications Systems*, 2002.

[76] B. Melander, M. Bjorkman, and P. Gunningberg. A new end-to-end probing and analysis method for estimating bandwidth bottlenecks. In *Global Telecommunications Conference, 2000. GLOBECOM '00. IEEE*, volume 1, pages 415–420 vol.1, 2000.

[77] Melanie Mitchell. *An introduction to genetic algorithms*. MIT press, 1998.

[78] J. Moy. RFC 2328 : OSPF Version 2. Technical report, IETF, 1998.

[79] J. Moy. RFC 7348 : Virtual eXtensible Local Area Network (VXLAN) : A Framework for Overlaying Virtualized Layer 2 Networks over Layer 3 Networks. Technical report, 2014.

[80] J.T. Moy. *OSPF, Anatomy of an Internet Routing Protocol*. Addison-Wesley, 1998.

[81] E. Mulyana and U. Killat. Optimizing ip networks for uncertain demands using outbound traffic constraints. In *INOC'2005*, pages 695–701, 2005.

[82] Walter Murray and Kien-Ming Ng. An algorithm for nonlinear optimization problems with binary variables. *Computational Optimization and Applications*, 47(2) :257–288, 2010.

[83] Katta G Murty. *Linear programming*, volume 57. Wiley New York, 1983.

[84] A. Nucci, R. Cruz, N. Taft, and C. Diot. Design of IGP link weight changes for estimation of traffic matrices. In *IEEE Infocom*, Hong Kong, March 2004.

[85] E. Nygren, R. K. Sitaraman, and J. Sun. The akamai network : A platform for high-performance internet applications. *ACM SIGOPS Operating Systems Review*, 44(3), July 2010.

[86] D. Oran. RFC 1142 : OSI IS-IS Intra-domain Routing Protocol. Technical report, IETF, 1990.

[87] Jay Painter and Ethan A. Merritt. Optimal description of a protein structure in terms of multiple groups undergoing TLS motion. *Acta Crystallographica Section D*, 62(4) :439–450, Apr 2006.

[88] Christos H Papadimitriou and Kenneth Steiglitz. *Combinatorial optimization : algorithms and complexity*. Courier Corporation, 1998.

[89] K. Papagiannaki, N. Taft, and A. Lakhina. A distributed approach to measure IP traffic matrices. In *Internet Measurement Conference*, pages 161–174, 2004.

[90] PanosM. Pardalos. Continuous approaches to discrete optimization problems. In G. Di Pillo and F. Giannessi, editors, *Nonlinear Optimization and Applications*, pages 313–325. Springer US, 1996.

[91] V. Paxson. End-to-end routing behavior in the internet. In *in Proc. ACM SIGCOMM'96*, pages 25–38, Stanford, CA, USA, August 1996.

[92] D. Pendarakis, S. Shi, D. Verma, and M. Waldvogel. Almi : An application level multicast infrastructure. In *Proc of the 3rd USNIX Symposium on Internet Technologies and Systems (USITS)*, San Francisco, CA, USA, March 2001.

[93] L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. In *in Proceedings of the 3rd ACM Workshop on Hot Topics in Networks (HotNets-III)*, November 2004.

[94] Ribiere G. Polak, E. Note sur la convergence de méthodes de directions conjuguées. *ESAIM : Mathematical Modelling and Numerical Analysis - Modélisation Mathématique et Analyse Numérique*, 3(R1) :35–43, 1969.

[95] J. Postel. User datagram protocol. RFC 768, Internet Engineering Task Force, August 1980.

[96] John Postel. Transmission control protocol. RFC 793, Internet Engineering Task Force, September 1981.

[97] Florian A. Potra and Stephen J. Wright. Interior-point methods. In *Society for Industrial and Applied Mathematics (SIAM.* SIAM, 2000.

[98] H. Rahul, M. Kasbekar, R. Sitaraman, and A. Berger. Towards realizing the performance and availability benefits of a global overlay network. In *Passive and Active Measurement Conference, Adelaide, Australia*, March 2006.

[99] J. B. Rosen. The gradient projection method for nonlinear programming. part i. linear constraints. *Journal of the Society for Industrial and Applied Mathematics*, 8(1) :pp. 181–217, 1960.

[100] A. Rowstron and P. Druschel. Pastry : Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In *In the Proceedings of the 18th IFIP/ACM International Conference on Distributed Systems Platforms (Middleware 2001)*, 2001.

[101] Stefan Savage, Andy Collins, Eric Hoffman, John Snell, and Thomas Anderson. The end-to-end effects of internet path selection. *SIGCOMM Comput. Commun. Rev.*, 29(4) :289–299, August 1999.

[102] R. Schollmeier. [16] a definition of peer-to-peer networking for the classification of peer-to-peer architectures and applications. In *Proceedings of the First International Conference on Peer-to-Peer Computing*, P2P '01, pages 101–, Washington, DC, USA, 2001. IEEE Computer Society.

[103] Ruud Schoonderwoerd, Owen Holl, Janet Bruten, and Leon Rothkrantz. Ant-based load balancing in telecommunications networks. *Adaptive Behavior*, 5 :169–207, 1996.

[104] Yevgeny Seldin, Csaba Szepesvári, Peter Auer, Montanuniversität Leoben, Yasin Abbasi-yadkori, Peter Deisenroth, Csaba Szepesvári, and Jan Peters. Evaluation and analysis of the performance of the exp3 algorithm in stochastic environments.

[105] R. K. Sitaraman, M. Kasbekar, W. Lichtenstein, and M. Jain. *Overlay Networks : An Akamai Perspective.* In Advanced Content Delivery, Streaming, and Cloud Services. John Wiley & Sons, 2014.

[106] A. Soule, A. Lakhina, N. Taft, K. Papagiannaki, K. Salamatian, A. Nucci, M. Crovella, and C. Diot. Traffic matrices : balancing measurements, inference and modeling. *SIGMETRICS Perform. Eval. Rev.*, 33(1) :362–373, 2005.

[107] A. Soule, K. Salamatian, A. Nucci, and N. Taft. Traffic matrix tracking using kalman filters. *SIGMETRICS Perform. Eval. Rev.*, 33(3) :24–31, 2005.

[108] N. Spring, R. Mahajan, D. Wetherall, and T. Anderson. Measuring isp topologies with rocketfuel. *Networking, IEEE/ACM Transactions on*, 12(1) :2 – 16, feb. 2004.

[109] I. Stoica, R. Morris, D. Karger, M.F. Kaashoek, and H. Balakrishnan. Chord : A scalable peer-to-peer lookup service for internet applications. In *SIGCOMM'01*, San Diego, California, USA., August 27-31 2001.

[110] R. Stone. Centertrack : An ip overlay network for tracking dos floods. In *in Proc. USENIX Security Symposium '00*, August 2000.

[111] Csaba Szepesvári. Shortest path discovery problems : A framework, algorithms and experimental results. In *AAAI*, pages 550–555, 2004.

[112] Geoff Huston Tony Bates, Philip Smith. Cidr report - http ://www.cidr-report.org/as2.0/. Technical report, CIDR, 2015.

[113] J. Touch, Y. Wang, L. Eggert, and G. Finn. A virtual internet architecture. Technical Report ISI-TR-2003-570, ISI, March 2003.

[114] H. Ümit. A column generation approach for igp weight setting problem. In *CoNEXT*, pages 294–295, Toulouse, France, 2005.

[115] J. Vallet and O. Brun. Adaptive routing in ip networks using snmp link counts. In *Teletraffic Congress (ITC), 2013 25th International*, pages 1–3, Sept 2013.

[116] J. Vallet and O. Brun. Online ospf weights optimization in ip networks. *Computer Networks*, 60(0) :1 – 12, 2014.

[117] Y. Vardi. Bayesian inference on network traffic using link count data. *Journal of the American Statistical Association*, 93(442) :573–576, June 1998.

[118] J. Wang, L. Lu, and A.A. Chien. Tolerating denial-of-service attacks using overlay networks - impact of overlay network topology. In *in Proc. First ACM Workshop on Survivable and Self-Regenerative Systems*, 2003.

[119] T Ye, H T Kaur, S Kalyanaraman, K S Vastola, and S Yadav. Dynamic optimization of ospf weights using online simulation. In *IEEE INFOCOM*, 2002.

[120] Y. Zhang. Abilene traffic matrices. http://www.cs.utexas.edu/~yzhang/research/AbileneTM/, 2004.

[121] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *ACM SIGMETRICS, San Diego, USA*, June 2003.

[122] Y. Zhang, M. Roughan, N. Duffield, and A. Greenberg. Fast accurate computation of large-scale ip traffic matrices from link loads. In *In ACM SIGMETRICS*, pages 206–217, 2003.

[123] B. Y. Zhao, L. Huang, J. Stribling, S.C. Rhea, A.D. Joseph, and J.D. Kubiatowicz. Tapestry : A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 2003.

[124] H. Zimmermann. Osi reference model–the iso model of architecture for open systems interconnection. *Communications, IEEE Transactions on*, 28(4) :425–432, Apr 1980.